AD NUMBER

AD911357

NEW LIMITATION CHANGE

TO

Approved for public release, distribution unlimited

FROM

Distribution authorized to U.S. Gov't. agencies only; Test and Evaluation; FEB 1972. Other requests shall be referred to Air Force Avionics Laboratory, Attn; AAM, Wright-Patterson AFB, OH 45433.

AUTHORITY

AFAL ltr, 7 Oct 1977

AFAL-TR-73-203
Volume III

AVIONICS PROCESSOR-CONTROLLER CONFIGURATION STUDY

APPENDIX B-VOLUME III

L. J. Koczela
Electronics Group of Rockwell International
Anaheim, California 92803

TECHNICAL REPORT AFAL-73-TR-203 VOL. III

AIR FORCE AVIONICS LABORATORY
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AVIONICS PROCESSOR-CONTROLLER CONFIGURATION STUDY

APPENDIX B-VOLUME III

L. J. Koczela

FOREWORD

This Final Engineering Report was prepared by the Electronics Group of Rockwell International, Anaheim, California. The work was accomplished under USAF Project 6090 entitled "Avionics Data Handling Technology", Task 01 entitled "Avionics Information Processing" and contract No. F33615-72-C-1973 entitled "Avionics Processor-Controller Configuration Study." The work was administered under the direction of Mr. J. E. Camp, Air Force Avionics Laboratory, AFAL/AAM, Wright-Patterson AFB, Ohio.

This report covers work conducted from 1 July 1972 to 30 June 1973 and was submitted by the author 30 April 1973.

This technical report has been reviewed and is approved for publication.

COLLIER S. KLINE
Colonel, USAF
Chief
System Avionics Division

ABSTRACT

This volume presents a detailed description of the Burroughs Multiprocessor. The descriptive material of the multiprocessor was scattered through several reports. The purpose of this volume is to extract the appropriate material from these reports and present the available material, upon which the study was based, in one unified report. This report is also being published as Autonetics internal report C72-812/201.

CONTENTS

ILLUSTRATIONS

TABLES

I. BURROUGHS COMPUTER DESCRIPTION

This appendix contains, for the most part, material extracted from various Burroughs reports (Ref B-1 through B-6). These reports provided descriptions of various portions of the Burroughs computer. It is the intent of this section to bring together some of this material into one unified document and also provide a basis for the Burroughs computer description that served as the basis for the study.

The Burroughs computer concept has been referred to by various acronyms in recent reports (Ref B-1 through B-5) such as the Interpreter Based System, Multiprocessor, and Aerospace Multiprocessor; in addition, the title of this study uses the acronym, avionics processor - controller. The term Burroughs multiprocessor or simply the multiprocessor will be used in place of these acronyms in this report.

A block diagram indicating the general structure of the Burroughs multiprocessor is given in Figure B-1. The basic modules or building blocks of the multiprocessor are:

1. Interpreters - Processing Elements consisting of arithmetic logic and alterable microprogram controls

2. SWI (Switch Interlock Unit) - Interconnection logic to allow interpreters to communicate with memories and devices

3. Memories - Storage elements for programs and data

4. Devices - interface elements between peripherals and the SWI

5. PSU (Port Select Unit) - May be used in place of the SWI for single interpreter systems

The Burroughs multiprocessor emphasizes two concepts (a) building block structure and (b) variable machine architecture achieved through microprogramming.

The basic building blocks listed above allow multiprocessors with different numbers of modules to be constructed to meet varying computational requirements. The multiprocessor designed for the Air Force allowed up to five Interpreters, eight Memories, and eight Devices.

Variable machine architecture is possible with the Burroughs multiprocessor by reloading the microprogram memory with routines. For example, it is possible to (a) emulate existing computers, (b) perform higher order language processing, and (c) process a problem optimized instruction set. Further, these could be performed in a multiprocessing manner.

The computer can operate as a true multiprocessor since any interpreter may access any memory or device module and multiple interpreters may be used simultaneously to process a computational task. Through the flexibility offered by variable machine architecture the interpreter can function as a CPU, as an I/O Processor, or as a device controller.

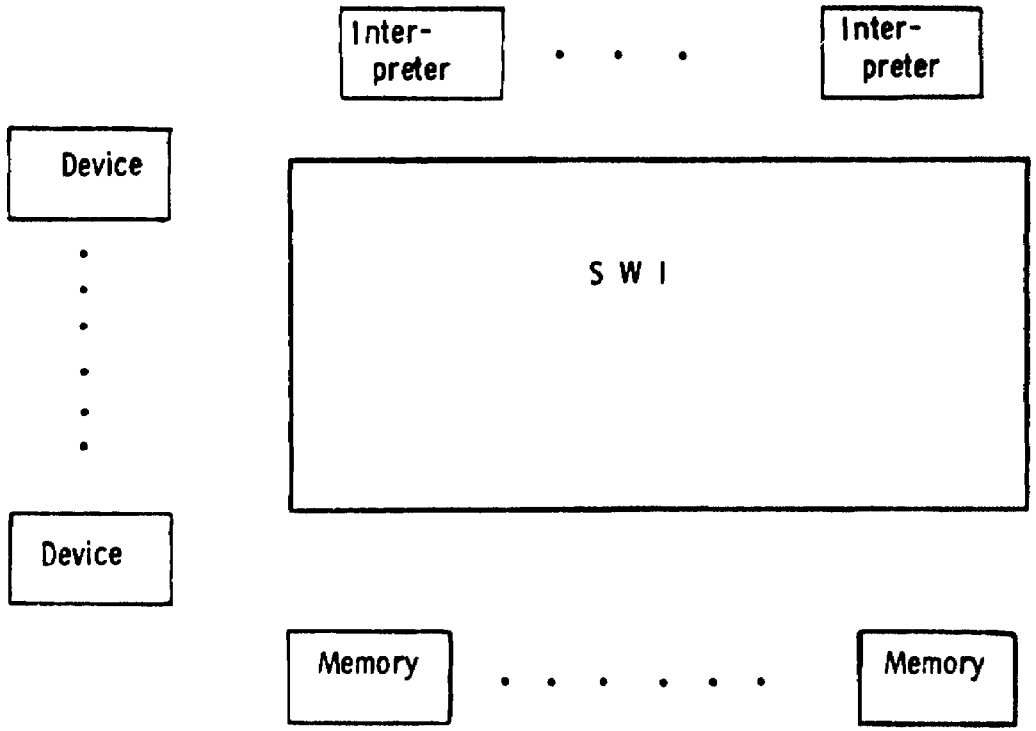


Figure B-1. Burroughs Multiprocessor

The Interpreter is a building block that can easily change its performance characteristics and can function as a CPU or an I/O module or as a device controller. It is a simple computer with primitive registers and combinatorial logic without the usual hardware control found in conventional computers. Initially the Interpreter is an uncommitted piece of hardware that is structured by stored logic contained in its microprogram memory.

A single Interpreter system can be mechanized using a port selection unit to interface the memories and devices.

2. INTERPRETER DESCRIPTION

A. GENERAL

The interpreter's functions are to:

1. Contain the microprogram memory.

2. Provide the timing and control for sequencing and controlling according to the microprogram memory.

3. Control the communication with external devices and memories.

4. Perform the logical and arithmetic operations required.

In order to accomplish these in a flexible manner Burroughs has defined a modular approach with the Interpreter consisting of submodules as follows (See Figure B-2):

1. Logic Unit - The circuitry associated with the arithmetic, shifting, and logic functions are contained in the Logic Unit. The data word length is expandable from 8 to 64 bits in 8 bit increments.

2. Control Unit - The Control Unit contains registers for conditional control and logic commands.

3. Memory Control Unit - The Memory Control Unit provides registers and control for memory (interpreter and main memory) addressing.

4. Microprogram Memory - This unit provides storage for the microprogram sequences. The unit could be implemented with ROM or RAM devices.

5. Nanomemory - The microcontrols for an Interpreter are supplied by the 54 bit wide Nanomemory. Most likely implementation of this is with the use of ROM. The particular nanoword is selected by the MPM word using the contained memory address.

B. LOGIC UNIT (LU)

Figure B-3 contains a detailed description of the data and control flow in the interpreter and Table B-1 identifies the control provided by the 54 bits in the nano-memory word. Reference to Figure B-3 and Table B-1 will aid in following the interpreter description given below.

One Logic Unit for each 8 bits of data word is required for each interpreter. The LU is composed of: the three A registers, a B register, an MBR register, adder, and barrel switch logic.

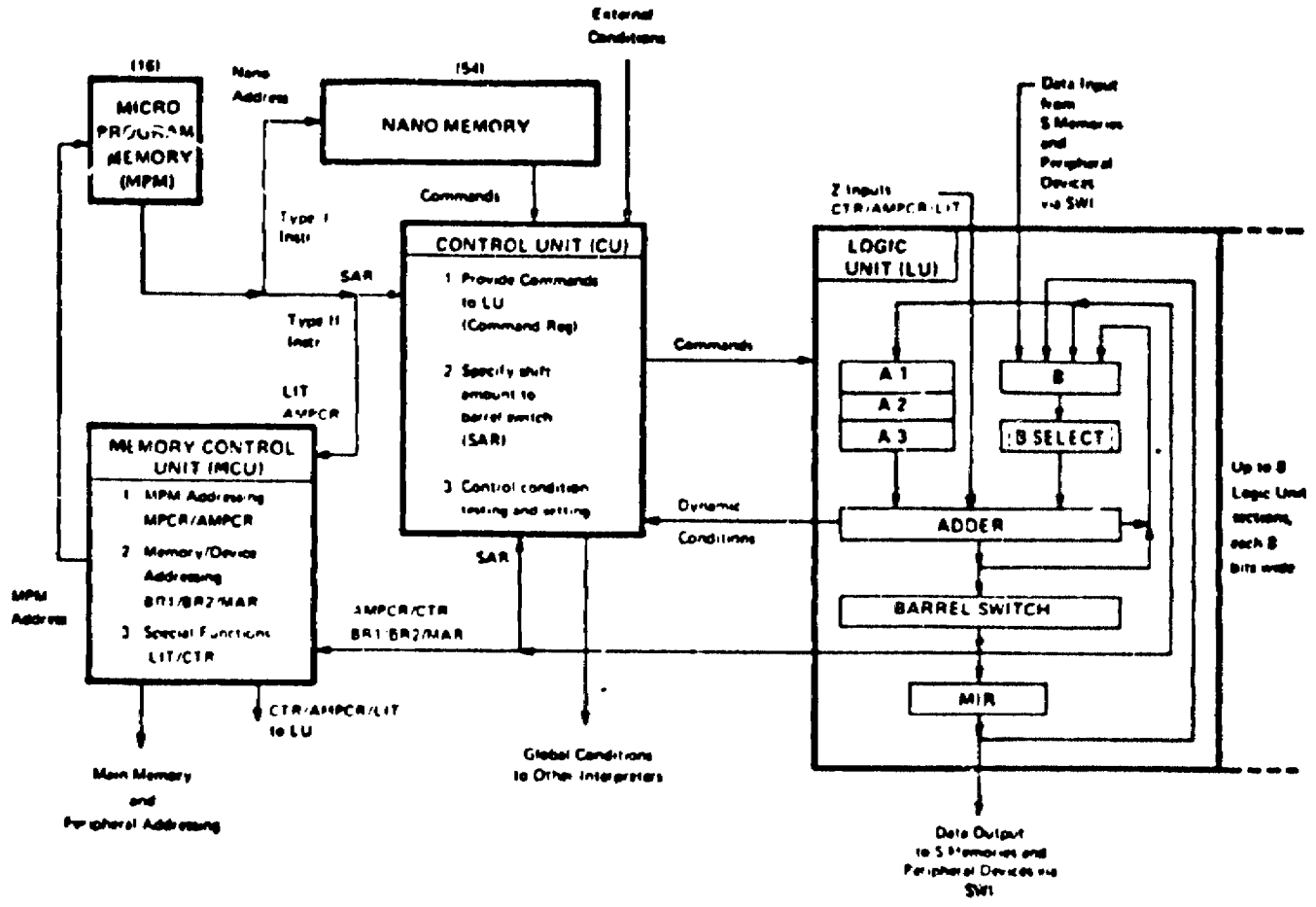
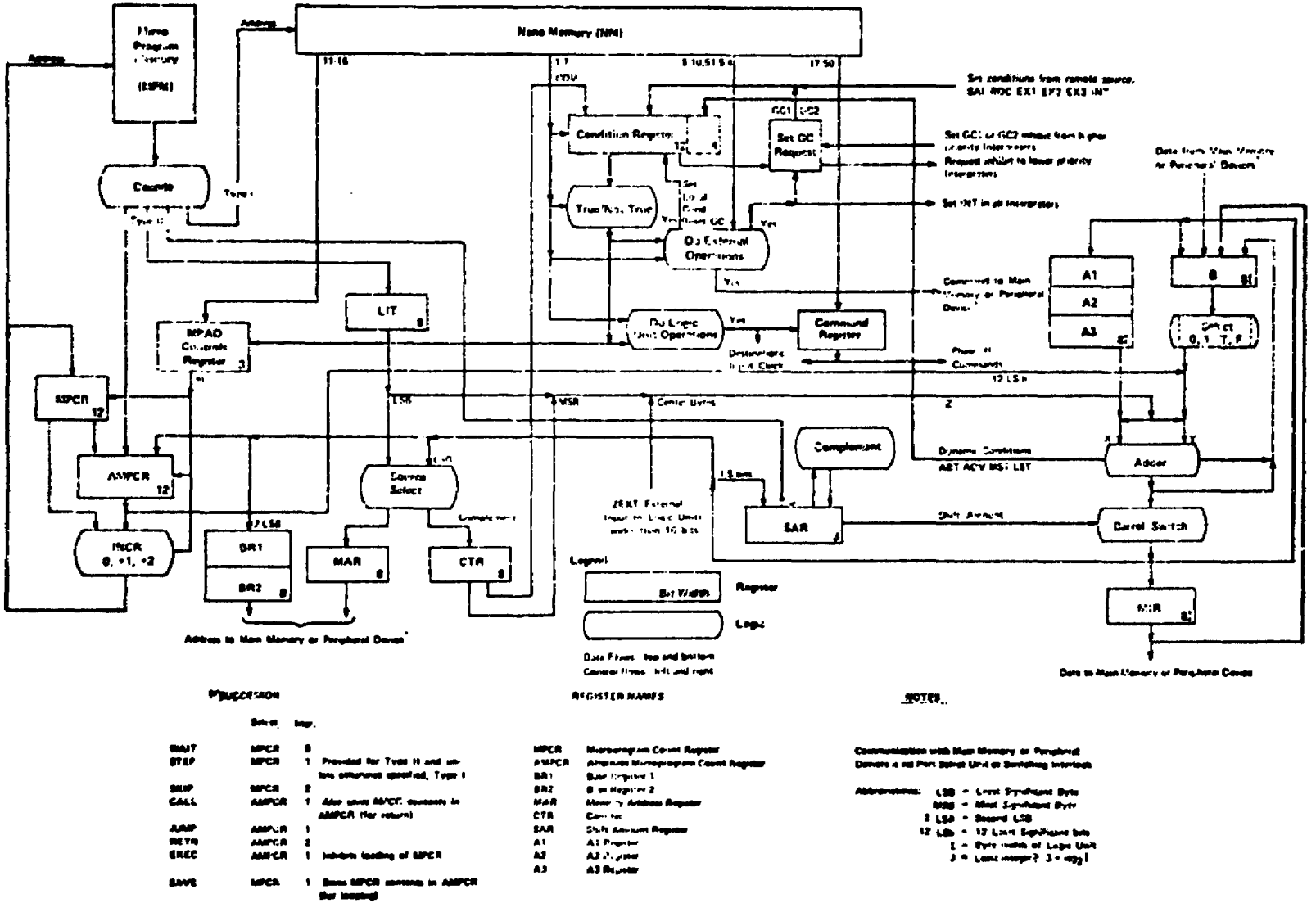


Figure B-2. Interpreter Block Diagram



SUCCESSION

	Select	Inst.	
WAIT	MPCR	0	
STEP	MPCR	1	Provided for Type II and unless otherwise specified, Type I
SKIP	MPCR	2	
CALL	AMPCR	1	Also saves MPCR contents in AMPCR (for return)
JUMP	AMPCR	1	
RETN	AMPCR	2	
EXEC	AMPCR	1	Initiates loading of MPCR
SAVE	MPCR	1	Saves MPCR contents in AMPCR (for looping)

REGISTER NAMES

MPCR	Microprogram Control Register
AMPCR	Alternate Microprogram Control Register
BR1	Base Register 1
BR2	Base Register 2
MAR	Memory Address Register
CTR	Counter
SAR	Shift Amount Register
A1	A1 Register
A2	A2 Register
A3	A3 Register

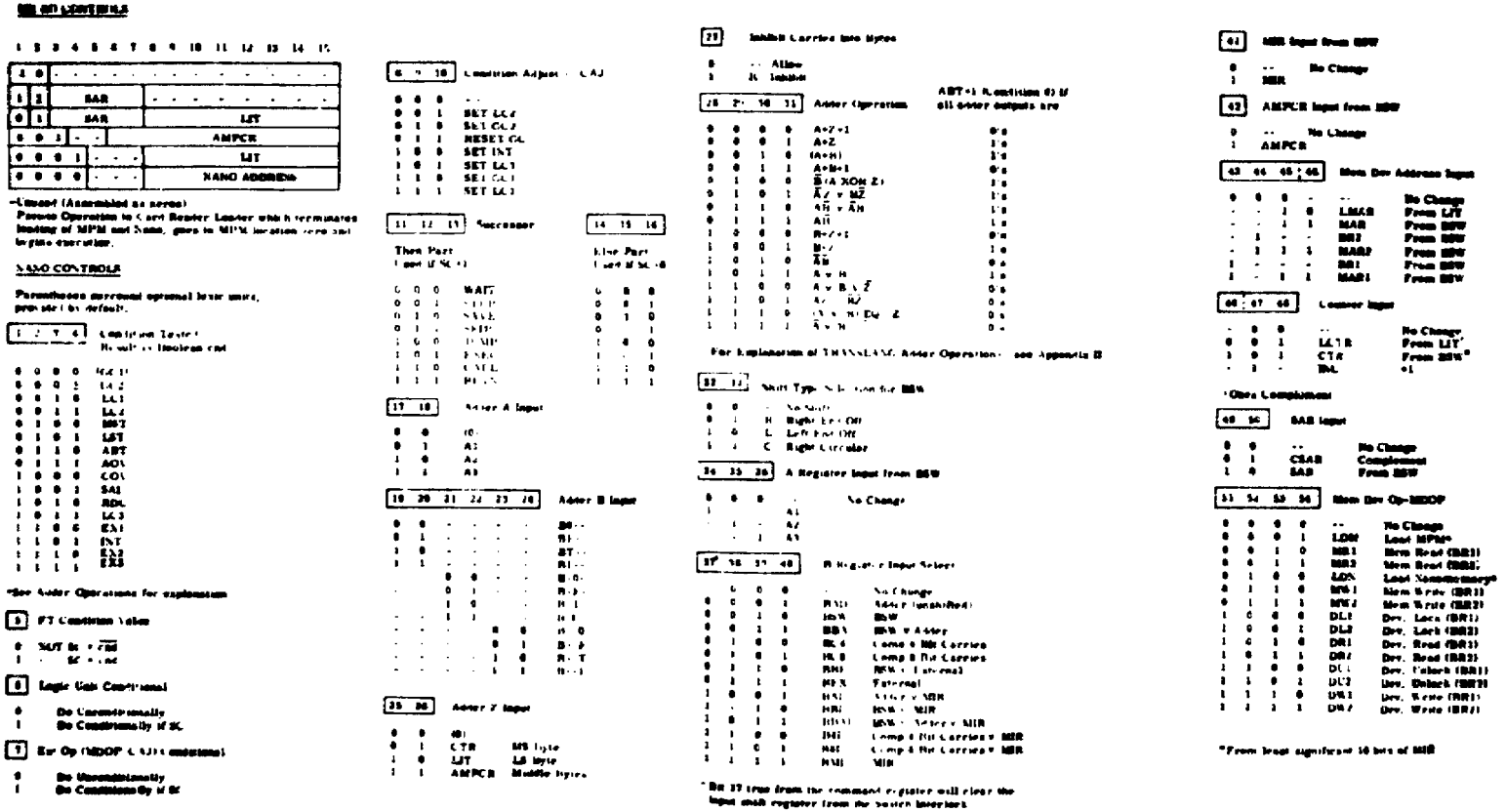
NOTES:

Communication with Main Memory or Peripheral Devices via Port Select Unit or Switching Interlock

Abbreviations:
- LSB = Least Significant Byte
- MSB = Most Significant Byte
- 2 LSB = Second LSB
- 12 LSb = 12 Least Significant bits
- 1 = Byte inputs of Logic Unit
- J = Least integer? $3 = \log_2 8$

Figure B-3. Interpreter Data and Control Flow

Table B-1. Detailed Nanobit Assignment



Registers A1, A2, and A3 are functionally identical. Each temporarily stores data and serves as a primary input to the adder. Selection gates permit the contents of any A register to be used as one of the inputs to the barrel switch.

The B register is a primary external interface (from the Switch Interlock). It serves as the second input to the adder and can also collect certain side effects of arithmetic operations. The B register may be loaded with any of the following (one per instruction):

1. The barrel switch output
2. The adder output
3. The data from the Switch Interlock
4. The MIR output
5. The carry complements (from the adder) of 4- or 8-bit groups with selected zeroes (for use in decimal arithmetic or character processing).
6. The barrel switch output ORed with 2, 3, or 4 above (within the B register)

The output of the B register has true/complement selection gates which are controlled in three separate sections: the most significant bit, the least significant bit, and all the remaining central bits. Each of these parts is controlled independently and may be either all ZEROs, all ONEs, the true contents or the complement (one's complement) of the respective bits of the B register.

The MIR buffers information being written to main memory or to a peripheral device. It is loaded from the barrel switch output and its output is sent to the Switch Interlock, or to the B register.

Inputs to the adder are from selection gates which allow various combinations of the A, B and Z inputs. The A input is from the A register output selection gates and the B input from the B register true/complement selection gates. The Z input is an external input to the LU and can be:

1. The 8-bit output of the counter of the MCU into the most significant 8 bits with all other bits being ZEROs.
2. The 8-bit output of the literal register of the MCU into the least significant 8 bits with all other bits being ZEROs.
3. The 12-bit output of the alternate microprogram count register (AMPCR) right justified into the middle 16 bits and the (wired) Interpreter number right justified in the remaining four bits of the middle 16 bits. All other bits are zeros.
4. All ZEROs.

Using various combinations of inputs to the selection gates, any two of the three inputs can be added together, or can be added together with an additional ONE added to the least significant bit. Also, all binary Boolean operations between the A and B and between the B and Z adder inputs and most of the binary Boolean operations between the A and Z adder inputs can be done.

The barrel switch is a matrix of gates that shifts a parallel input data word any number of places to the left or right, either end-off or end-around, in one clock time.

The output of the barrel switch is sent to:

1. The A registers (A1, A2, A3)

2. The B register

3. Memory Information Register (MIR)

4. Least significant 16 bits to MCU (registers BR1, BR2, MAR, AMPCR, LIT, CTR)

5. Least significant 5 bits to shift amount register (SAR) in the CU.

C. CONTROL UNIT (CU)

One CU is required for each Interpreter. Major sections of this unit are: the shift amount register (SAR), the condition register, part of the control register (CR), the MPM content decoding, and the clock control.

The functions of the SAR and its associated logic are:

1. To load shift amounts into the SAR to be used in the shifting operations.

2. To generate the required controls for the barrel switch shift operation indicated by the controls from the Nanomemory.

3. To generate the "word length complement" of the SAR contents, where the "complement" is defined as the amount that will restore the bits of a word to their original position after an end-around shift of N followed by an end-around of the "complement" of N.

The condition register section of the CU performs four major functions:

1. Stores 12 resettable condition bits in the condition registers. The 12 bits of the condition register are used as error indicators, interrupts, status indicators and lockout indicators.

2. Selects 1 of 16 condition bits (12 from the register and 4 generated during the present clock time in the Logic Unit) for use in performing conditional operations.

3. Decodes bits from the Nanomemory for resetting, setting, or requesting the setting of certain bits in the condition register.

4. Resolves priority between Interpreters in the setting of global condition (GC) bits.

The control register is a register that stores 38 of the 54 control signals from the Nanomemory that are used in the LU, CU, and MCU for controlling the execution phase of a microinstruction. Twelve of the 38 outputs from the Nanomemory are stored in the CU. Four of the other 38 Nanomemory outputs are controls to the Switch Interlock and are stored there. The other 22 of the 38 Nanomemory outputs are stored in a part of the control register physically located in the Nanomemory.

The MPM content decoding determines (based upon the first four bits of the MPM) whether the MPM output is to be used as a Type I instruction (Nanomemory address) or as a Type II instruction (literal). Several decoding options are available.

D. MEMORY CONTROL UNIT (MCU)

One MCU is required for an Interpreter, but a second MCU may be added to provide additional memory addressing capability. This unit has three major sections:

1. The microprogram address section contains the microprogram count register (MPCR), the alternate microprogram count register (AMPCR), the incrementer, the microprogram address control register, and associated control logic. The output of the incrementer addresses the MPM for the sequencing of the microinstructions. The AMPCR contents are also used as one of the Z inputs to the adder in the LU.

2. The memory/device address section contains the main memory address register (MAR), base registers one and two (BR1, BR2), the base register output selection gates, and the associated control logic.

3. The Z register section contains registers which are two of the Z inputs to the LU adder: a loadable counter (CTR), the literal register (LIT), selection gates for the input to the memory address register and the loadable counter and their associated control logic.

E. NANOMEMORY (N MEMORY)

The Interpreter is controlled by the output of the 54-bit wide Nanomemory which may be implemented with a read/write memory, a read-only memory, wired logic, or a combination of the three.

Each of the 54 bits represents a unique enable line for the gates and flip-flops within the LU, the CU, and the MCU. Each Nanomemory word represents a microinstruction that is executed by the simultaneous presentation of a specific enable pattern for the 54 outputs, represented by corresponding ONEs and ZEROs in its word. The definition of these bits is presented in the microprogramming section.

A unique feature of the Interpreter Based System with its separate Nanomemory and Microprogram Memory is that the explicit enable lines for each microinstruction need be stored in the Nanomemory only once (regardless of the number of times that a specific microinstruction is needed in a program). To accomplish this saving in memory, the Microprogram Memory (MPM) contains the address in the Nanomemory where the explicit ONEs and ZEROs are stored that are needed to execute that instruction type rather than the full microinstruction. Thus, several microprogram sequences which use the same microinstruction (e.g., transfer A to B) need only store in the Microporgram Memory the address of the Nanomemory word containing that microinstruction.

F. MICROPROGRAM MEMORY (MPM)

Each Interpreter requires a source of microprogram instructions to define the operation of the Interpreter.

Two possible solutions for providing this source of microprogram instructions are listed below:

1. A semiconductor MPM. This memory can be a read-only memory (ROM) if the Interpreter is to be dedicated to the function defined by the ROM. A read-write memory can be used for experimental purposes or when the function of the Interpreter might be changed, such as reconfiguration in a multiple Interpreter system. In this instance, the system could afford to wait while the MPM was reloaded from a remote microprogram store accessed via the Switch Interlock.

2. A buffer into a slower-speed, wider-word memory.

Loading of the MPM and NM can be from an external source if both are read-write types. This external source can be the AGE or operating memory.

3. INTERPRETER OPERATION

A. GENERAL

A unique feature of the Interpreter Based System is the utilization of stored logic in M and N memories and uncommitted hardware logic to form firmware control that is exercised to a more primitive logic level than in conventional microprogrammed central processors, being read at every clock time and offering more parallelism in its greater word length. This firmware, in essence, commits the hardware logic of the system to function in a specific fashion. The highest level of instruction used in an Interpreter is the S instruction which corresponds to a standard machine instruction and is stored in Data/Program (S) Memory. In a typical application, a starter set of microinstructions is accessed which causes the first word of the first S instruction of the program to be called into the interpreter. An analysis group of microinstructions causes the op-code of the S instruction to be converted into the address of the first of a string of microninstructions (m-string). That particular m-string provides all of the control necessary for the execution of the S instruction including the calling of any additional memory words that might be required to describe the full S instruction. The m-string terminates by transferring control to the m-string that calls the next S instruction.

Each microinstruction requires a single clock cycle for its execution. The 16 bit microinstruction either contains literal data (Type II microinstruction) or it contains the address of a 54 bit word called the nanoinstruction in the Nanomemory of the Interpreter which when read provides the information to produce a set of 54 logical levels (Type I microinstruction). In either case, the 54 logical levels control the hardware logic in the Interpreter which provide the desired function. The control signals, or enables, for the hardware logic which are exercised by the nanoinstructions of the N memory are summarized in Table B-1. Figure B-3 presents the interpreter data and control flow.

During each clock period, a 16 bit microinstruction is read from the MPM. The first four bits of this microinstruction indicate which of two types of instruction it is. If it is a Type I instruction, the remaining bits of the MPM word specify a Nanomemory address to be accessed. The Nanomemory is then initiated and its output, a set of 54 bits, provides the control functions as indicated in the listing in Table B-1.

If the microinstruction is Type II, the remaining bits of the MPM word are stored into one or two registers: namely, the SAR, LIT, SAR and LIT, or the AMPCR. The determination of which registers are to be loaded is specified by the first four bits of the MPM word. The Nanomemory is not accessed during a Type II operation.

Each Type I microinstruction has two parts (or phases). The first fetches the instruction from the MPM and Nanomemory and the second executes the fetched instruction. Figure B-4 illustrates these two basic phases of each Type I microinstruction.

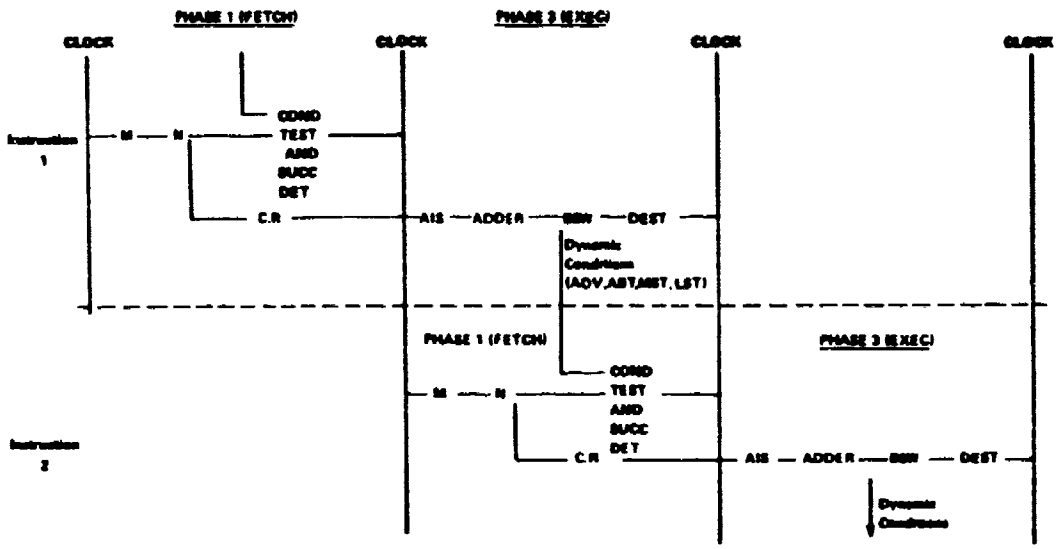


Figure B-4. Timing Analysis Type I Instructions

The fetch phase involves: MPM accessing, Nanomemory accessing, condition testing, selection of controls for the next instruction (successor) address computation, and, in parallel, loading the control register for the execution of the microinstruction. A fetch phase occurs for every Type I microinstruction and requires one clock time. Since it always overlaps the execution phase of a prior Type I microinstruction, the performance of each microinstruction requires effectively one clock interval. (Subject to the conditions listed below.)

The execution phase also requires one clock time and always overlaps the fetch phase of the next Type I instruction. The control signals for the execution phase are from the output of the control register and have two parts: signals specifying the logic unit operation (adder input selection, adder function, barrel switch shifting, etc.) and signals specifying the destination register(s) loading (i.e. clock enables). Both sets of these controls apply continuously from the start to the end of the phase; however, the destination registers are not changed until the occurrence of the clock pulse which signals the end of the execution phase and which simultaneously reloads the control register for the execution of a new logic unit operation. The completion of the execution phase (i.e., the destination register(s) loading), may be delayed or suspended for one or more clock times.

Suspended execution phase is the name given to an execution phase clock time whose logic unit operation has been and continues to be performed but whose destination register loading is postponed for one or more clock periods. The register loading part of an execution phase depends on the subsequent microinstructions which follow the Type I instruction.

This suspended execution phase can occur for three primary reasons. The first and most frequent occurrence is when the next instruction from the MPM is a Type II instruction. This Type II instruction is executed during the same clock time it is fetched and the execution of the Type I instruction in progress is held in this suspended execution phase until the next clock interval. This allows the fetch phase of the next microinstruction (if it is a Type I) to have an execution phase to overlap. This provides condition bits (generated dynamically during the execution phase of a microinstruction) that can be tested during the fetch phase of the next Type I microinstruction.

The second reason for the occurrence of a suspended execution phase is due to the existence of conditional logic unit operations. A Type I microinstruction which does not contain a conditional logical operation always has a fetch phase and an execution phase. However, a Type I microinstruction which does contain a conditional logical operation falls into either of two categories: if the condition is met, both the fetch phase and execution phase are required for the execution to be completed; if the condition is not met, only the fetch phase is required for the completed execution to occur. However, even though the execution phase of a conditional Type I microinstruction is ignored, the fetch phase of the next Type I microinstruction must have an execution phase to overlap in order to have values for dynamic conditions that may be Type I and not clocking the execution phase that is to be disregarded into the control register.

The other reason for a suspended execution phase is for use during the loading of the MPM and Nanomemory.

The sequencing of Type I microprogram instructions is controlled by the following procedure: The MPM addresses the nanomemory which provides information to the condition testing logic indicating which condition is to be tested. The condition testing logic provides a True/False signal to the successor selection logic which selects between the three True and three False successor bits (also from the Nanomemory). The three selected bits (True/False) provide eight possible successor command combinations discussed later. A Type II microinstruction has an implicit STEP successor.

B. CONDITION OPERATIVES

Each N instruction performs a test on the Boolean value of one condition or its complement. The test of a condition is used to allow conditional microinstruction successor selection, conditional logic unit operations, and/or conditional external operations. With the exception of the two global condition bits, testing a condition bit causes the bit to be reset. The least and most significant bits out of the adder, the adder overflow, and the adder bit transmit are levels and not condition bits. The conditions that may be tested (Table B-2) are the following:

SAI Switch Interlock Accepts Information

Following memory or device operation, indicates that connection to the addressed memory or device is completed through the switch interlock.

Table B-2. Set and Reset of Conditions

Bit	Set	Reset
AOV	Dynamic Adder State - (Overflow)	#
ABT	Dynamic Adder State - (Adder bit transmit)	#
LST	Dynamic Adder State - (Least Significant Bit of Adder Outputs)	#
MST	Dynamic Adder State - (Most Significant Bit of Adder Output)	#
COV	Overflow when Counter is Incremented	Reset by loading counter or by testing
GC1	Set GC1 providing no other Interpreter has GC1 set, or no higher priority Interpreter is concurrently doing SET CG1	RESET GC
GC2	SET GC2 similar to GC1	RESET GC
INT	Set INT executed in any Interpreter	Reset by testing*
LC1	SET LC1	Reset by testing
LC2	SET LC2	Reset by testing
LC3	SET LC3	Reset by testing
RDC	By memory at completion of memory or device read	Reset by testing
SA1	By switch interlock or PSU when data received from MAR and MIR	Reset by testing
EX1	By requests from devices	Rest by testing*
EX2	By requests from devices	Reset by testing*
EX3	By requests from devices	Reset by testing*

#Recomputed each clock time

*In local Interpreter only

RDC Read Complete, or Requested Device Completes

Following memory read or device read by request, indicates that data will be available for entry to B in the next clock. Following device write by request, indicates completion.

COV Counter Overflow

Following or concurrent with increment counter INC, indicates counter is overflowing or has already overflowed from all ones (255) to all zeros.

LC1 Local Condition 1

Tests and resets local Boolean condition bit LC1.

LC2 Local Conditions 2 and 3

LC3 Same as LC1

GC1 Global Conditions 1 and 2

GC2 Tests but does not reset global condition bit GC1. See the description of the set and reset operation for further explanation of global condition bits.

INT Inter-Interpreter Interrupt

Tests and resets the local copy of the inter-Interpreter interrupt.

EX1 External Conditions 1, 2 and 3

EX2 Test and reset interrupts (usually the OR of interrupts from several
EX3 devices) from external devices (local copy).

The following four logic unit conditions are dynamic and indicate the result output from the adder in the phase 3 commands from the previous instruction which had logic unit operation, and using the current values of the adder inputs. These conditions are sustained until execution of another instruction involving the logic unit, and may be tested by that instruction. A Type II instruction loading the LIT or AMPCR may change the value of an adder input selected and hence change the value of any of these conditions.

AOV Adder Overflow

Results from an adder operation with carry out of the most significant end of the adder.

LST Least significant

State of the least significant bit of the adder output.

MST Most significant

State of the most significant bit of the adder output.

ABT Adder bit transmit

This condition is true (one) if and only if the adder output is all ones.

The set and reset operations are used to set and reset condition bits. The inter-Interpreter interrupt INT, is used for communication (to signal) all Interpreters of a multiprocessing system. The global conditions, GC1 and GC2, are used as Boolean semaphores to guarantee mutual exclusion for critical sections of m-program and to prevent simultaneous access to shared data. The local condition bits are Boolean variables local to each Interpreter. The INT and local condition bits are reset (within the local Interpreter only) by testing. The explicit test and reset operations follow.

SET INT Interrupt Interpreters

Causes the interrupt bit to be set in all Interpreters. Each Interpreter resets its own bit by testing it. Setting occurs after testing should both occur in the same N-instruction.

SET LC1 Set the first local condition bit

Causes the setting of the LC1 bit in the condition register, setting occurs after testing should both occur in the same N-instruction, both set and test of LC1 occur in Phase 1.

SET LC2 Set the second local condition bit

Same as for LC1 replacing LC1 by LC2.

SET LC3 Set third local condition bit

Same as for LC1 replacing LC1 by LC3.

SET GC1 Set first global condition bit request

Requests that the GC1 bit in the requesting Interpreter be set if a GC1 bit is not already set in another Interpreter or is not being set simultaneously by a higher priority Interpreter. For all Interpreters in a multiprocessing system at most one will have GC1 set. GC1 is set at the end of the phase after Phase 1 if no conflict occurs. A request lasts for one clock.

SET GC2 Set second global condition bit request

Same as for GC1 replacing GC1 by GC2.

RESET GC Resets the global condition bits

Causes GC1 and GC2 to be reset in the issuing Interpreter.

C. MICROINSTRUCTION SEQUENCING

Each N instruction performs a test on the Boolean value of one condition on its complement. If the result is true the successor for this condition is used to determine the next N-instruction. Otherwise the successor for the not condition is used to determine the next M-instruction address.

Successor: The successor commands are as follows:

1. Step to the Next Instruction in Sequence (STEP):

   The next instruction address is the content of the MPCR plus one. The MPCR content will be replaced by the next instruction address.

2. Skip the Next Instruction (SKIP):

   This operation permits one instruction conditional branches without an explicit address specification. The next instruction address is the content of the MPCR plus two. The MPCR content will be replaced by the next instruction address.

3. Repeat the Instruction (WAIT):

   This operation permits the repeated execution until the value of the condition changes. The next instruction address is the content of the MPCR. The MPCR content will be unchanged.

4. Save Loop Address (SAVE):

   This operation is usually performed just before entering the first iteration of a loop. It causes the address of the current instruction to be saved in the AMPCR so that jumps can be made later to the current instruction address plus one. The AMPCR is replaced by the contents of the MPCR. The next MPM instruction address is the content of the MPCR plus one. The MPCR will be replaced by the next instruction address.

5. Execute an Instruction Out of Sequence (EXEC):

   This operation permits the instruction named in AMPCR plus one to be executed without changes to either the MPCR or AMPCR. As an example, this operation can be used for address table lookup if the named instruction is an AMPCR literal. The AMPCR may change as indicated by the executed instruction.

6. Call a Procedure (CALL):

   This operation causes a jump to the routine specified in AMPCR plus one with the current position saved for later return. The AMPCR content will be replaced by the MPCR content. The MPCR content will be replaced by the next instruction address (AMPCR plus one).

7. Jump (JUMP):

   This operation permits transfer of control to the instruction named in the AMPCR plus one. This address may be a computed address loaded from the BSW output or an address constant from a microinstruction. This may be used to go to the head of a loop or to the return position for a procedure call. The next instruction address is the content of the AMPCR plus one. The MPCR content will be replaced by the next instruction address.

8. Return (RETN):

This operation provides for an alternative jump address by making the next instruction address be the content of the AMPCR plus two. The MPCR content will be replaced by the next instruction address.

The particular chosen successor command then provides controls used in the selection (MPCR/AMPCR) and incrementing logic which generates the next MPM address. Except for the EXEC command, the MPCR is loaded with this MPM address. Table B-3 summarizes the MPM addressing.

Table B-3. Microprogram Memory Addressing

Successor Command	Successor M-Instruction Address	Next Content of MPCR Will Be	Next Content of AMPCR Will Be
WAIT	MPCR	MPCR	*
STEP	MPCR+1	MPCR+1	*
SKIP	MPCR+2	MPCR+2	*
SAVE	MPCR+1	MPCR+1	MPCR
CALL	AMPCR+1	AMPCR+1	MPCR
EXEC	AMPCR+1	MPCR *	*
JUMP	AMPCR+1	AMPCR+1	*
RETN	AMPCR+2	AMPCR+2	*

*Not changed by successor specification

D. LOGIC UNIT OPERATIONS

1. Adder Operations

Inputs to the adder are from selection gates which allow various combinations of the A, B, and Z inputs. The A input is from the A register output selection gates and the B input from the B register true/complement selection gates. The Z input is an external input to the LU and can be:

1. The 8-bit output of the counter of the MCU into the most significant 8 bits with all other bits being ZEROs.

2. The 8-bit output of the literal register of the MCU into the least significant 8 bits with all other bits being ZEROs.

3. The 12-bit output of the alternate microprogram count register (AMPCR) right justified into the middle 16 bits and the (wired) Interpreter number right justified in the remaining four bits of the middle 16 bits. All other bits are zeros.

4. All ZEROs.

Using various combinations of inputs to the selection gates, any two of the three inputs can be added together, or can be added together with an additional ONE added to the least significant bit. Also, all binary Boolean operations between the A and B and between the B and Z adder inputs and most of the binary Boolean operations between the A and Z adder inputs can be done.

Table B-4 summarizes the adder arithmetic and logical operations that may be specified using TRANSLANG which is a microtranslator that produces micro and nano instruction from symbolic instructions. The following notes apply to this table:

1. A Register Selection	A	A1 A2 A3
	A0	All ZEROS
2. B Register Selection:	B	Any B Register Select option
	$\overline{B}$	ONES complement (by TRANSLANG) of the specified B Register Select option
	0	All ZEROS
	1	ALL ONES
3. Z Register Selection	Z	CTR | LIT | AMPCR
	0	All ZEROS
4. Inhibit 8 Bit Carry:	0	All carry into bytes
	1	Inhibit carry into bytes
5. Adder Operation		As specified in Microprogramming chart, Table B-1

2. Shifting

There are four operations causing shifting, one of which is selected each time an adder operator is used.

R Right end-off shift by amount in SAR, filled with left zeros.

L Left end-off by word length complement of amount in SAR, filled with right zeros.

C Circular right end-around shift by amount in SAR.

No shift

Table B-4. Adder Operations

Adder Operation	Result Form	Arithmetic Operations — Register Select A[1]	B[2]	Z[3]	IC8[4]	ADDOP[5]	ABT Is True If Result is All
A ADD B		A	B	0	0	2	Ones
A ADD Z	R - S	A	0	Z	0	1	Ones
B ADD Z		0	B	Z	0	9	Ones
A ADL B		A	B	0	0	3	Zeros
A ADL Z	R·S·1	A	0	Z	0	0	Zeros
B ADL Z		0	B	Z	0	8	Zeros
A CAD B	R - S	A	B	0	1	2	Ones
A CAD Z	Without	A	0	Z	1	1	Ones
B CAD Z	Carry	0	B	Z	1	9	Ones
DEC A		A	1	0	0	2	Ones
DEC B	R - 1			Invalid			
DEC Z		0	1	Z	0	9	Ones
0		0	0	0	0	2	Never
1		0	0	0	0	3	Never

Adder Operation	Result Form	MONADIC Logical Operations — Register Select A[1]	B[2]	Z[3]	ADDOP[5]	ABT Is True If Result is All
A		A	0	0	2	Ones
B	R	0	B	0	2	Ones
Z		0	0	Z	1	Ones
NOT A		A	0	0	15	Zeros
NOT B	$\overline{R}$	0	B	0	10	Zeros
NOT Z		0	0	Z	12	Zeros

Adder Operation	Result Form	DYADIC Logical Operations — Register Select A[1]	B[2]	Z[3]	ADDOP[5]	ABT is True If Result is All
A AND B		A	$\overline{B}$	0	7	Ones
A AND Z	R∧S	A	1	Z	13	Zeros
B AND Z		0	$\overline{B}$	Z	4	Ones
A NIM B		A	B	0	7	Ones
A NIM Z	R∧$\overline{S}$			INVALID		
B NIM Z		0	$\overline{B}$	Z	13	Zeros

Table B-4. (Cont)

Adder Operation	Result Form	DYADIC Logical Operations (Cont) Register Select A[1]	B[2]	Z[3]	ADDOP[5]	ABT is True If Result is All
A NRI B		A	$\overline{B}$	0	10	Zeros
A NRI Z	$\overline{R} \wedge S$	A	0	Z	5	Ones
B NRI Z		0	B	Z	4	Ones
A NOR B		A	B	0	10	Zeros
A NOR Z	$\overline{R} \wedge \overline{S}$	INVALID				
B NOR Z		0	B	Z	13	Zeros
A XOR B		A	B	0	6	Ones
Z XOR Z	$(R \wedge \overline{S}) \vee (\overline{R} \wedge S)$	A	0	Z	4	Ones
B XOR Z		0	$\overline{B}$	Z	14	Zeros
A EQV B		A	$\overline{B}$	0	6	Ones
A EQV Z	$(R \wedge S) \vee (\overline{R} \wedge \overline{S})$	A	0	Z	14	Zeros
B EQV Z		0	B	Z	14	Zeros
A NAN B		A	$\overline{B}$	0	15	Zeros
A NAN Z	$\overline{R} \vee \overline{S}$	A	1	Z	5	Ones
B NAN Z		0	$\overline{B}$	Z	12	Zeros
A IMP B		A	B	0	15	Zeros
A IMP Z	$\overline{R} \vee S$	INVALID				
B IMP Z		0	$\overline{B}$	Z	5	Ones
A OR B		A	B	0	11	Ones
A OR Z	$R \vee S$	INVALID				
B OR Z		0	B	Z	5	Ones
A RIM B		A	$\overline{B}$	0	11	Ones
A RIM Z	$R \vee \overline{S}$	A	0	Z	12	Zeros
B RIM Z		0	B	Z	12	Zeros

Adder Operation	TRIADIC Logical Operations ADDOP[5]	Result	ABT is True If Result is All
TRY1 A, B, Z	4	$\overline{B}$(A XOR Z)	Ones
TRY2 A, B, Z	5	$\overline{A}$ Z B $\overline{Z}$	Ones
TRY3 A, B, Z	12	A B $\overline{Z}$	Zeros
TRY4 A, B, Z	13	A Z $\overline{B}$ $\overline{Z}$	Zeros
TRY5 A, B, Z	14	(A B) EQV Z	Zeros

3. Data Loading

The principal data source is the barrel switch output. It is the only source for loading A1, A2, A3, MIR, BR1 and BR2. It provides one source for loading B, CTR, MAR, SAR and AMPCR. These reserved words are also the register names. The bits used in these transfers are indicated below.

Destination Register	Barrel Switch Output Source Bits
A1	All
A2	All
A3	All
B	All
MIR	All
BR1	2nd least significant byte
BR2	2nd least significant byte
MAR	least significant byte
CTR	least significant byte (ones complement)
SAR	least significant bits
AMPCR	least significant 12 bits

The B, MAR, CTR, SAR and AMPCR registers may have other inputs as well:

B Register - (B)

BSW The barrel switch output is placed into B

BEX Data from the external source is placed into B

BAD The adder output (unshifted) is placed in the B register

BC4 The duplicated complement of the 4-bit carries with zero fill is placed in the B register

BC8 The duplicated complement of the 8-bit carries with zero fill is placed in the B register

BMI The MIR content is placed in the B register independent of any concurrent change to the MIR

BBE The barrel switch output ORed with the data from the external source is placed in the B register

BBA The barrel switch output ORed with the adder output is placed in the B register.

BBI The barrel switch output ORed with the MIR content is placed in the B register independent of any concurrent change to the MIR.

BAI The adder output ORed with the MIR content

BBAI The barrel switch output ORed with the added output ORed with the MIR content

B4I The complement of the 4 bit carries ORed with MIR

B8I The complement of the 8 bit carries ORed with MIR

Memory Address Register - (MAR)

LMAR The literal register content is placed in MAR

Counter - (CTR)

LCTR The ones complement of the literal register content is placed in CTR

INC Increment Counter by 1

Shift Amount Register - (SAR)

CSAR Complement prior content of SAR

The Alternative Micro Program Count Register AMPCR may during the same clock receive input from the MPCR if the MPAD CTLS register content was CALL or SAVE. The MPCR source takes precedence over the AMPCR specification as a destination.

The destination operators explicitly specify registers in which changes are to occur at the end of a logic unit operation.

4. External Operations

The external operations are functions which if explicitly present affect the operations external to the Interpreter logic. An external operation may be specified as either conditional or unconditional.

The memory or device operations are used to transfer data between the Interpreter and main S-memory or a peripheral device. Address source registers for those operations are the concatenation of either BR1 or BR2 with MAR, indicated respectively by MAR1 or MAR2. The MAR part is less significant. The explicit memory or device operations follow. If none is specified,then any memory or device operation in progress is continued and no new operation is initiated. Address or MIR change may terminate the operation.

MR1 Memory Read 1

Read data from S-memory address specified in MAR1

MR2 Memory Read 2

Read data from S-memory address specified in MAR2

MW1 Memory Write 1

Write data from MIR to S-memory address specified in MAR1

MW2 Memory Write 2

Write data from MIR to S-memory address specified in MAR2

ASR Status Request for highest priority locked device*

ASE Status Request for highest priority unlocked device*

DL1 Device Lock 1 Request*

Reserve the device or memory module named in MAR1 for use by this Interpreter

DL2 Device Lock 2 Request*

Reserve the device or memory module named in MAR2 for use by this Interpreter

DR1 Device Read 1

Read data from device name in MAR1

DR2 Device Read 2

Read data from device named in MAR2

DW1 Device Write 1

Write data from MIR to the device name in MAR1

DW2 Device Write 2

Write data from MIR to the device name in MAR2

DU1 Device Unlock 1

Release the locked device named in MAR1

DU2 Device Unlock 2

Release the locked device named in MAR2

*Systems with switch interlock use DL1 and DL2; systems with port select unit use ASR and ASE.

4. SWITCH INTERLOCK (SWI) DESCRIPTION

A. SWI Modules

The Switch Interlock functions are to:

1. Provide the interconnection of the interpreters with the memories and devices.

2. Provide the priority for the interpreters in the selection of devices and memories.

Connection between Interpreters and devices is by reservation with the Interpreter having exclusive use of the (locked) device until specifically released. Connection with a memory module is for the duration of a single data word exchange, but is maintained until some other module is requested or some other Interpreter requests that module.

In any such system it is desirable to keep the wires and logic in the crosspoints to a minimum, while still maintaining a specified transfer rate. One way of achieving this is by serial transmission of several partial words in parallel through the crosspoints. The Switch Interlock for the Burroughs Multiprocessor handles up to five Interpreters, eight memories and eight devices. The transmission paths through the Switch Interlock break the 32-bit data word into 4 - 8 bit bytes.

The SWI is mechanized with five modules; a block diagram indicating the structure of the SWI is given in Figure B-5. This diagram also shows the internal and external interface of the SWI. The five modules are:

1. Memory Device Control (MDC) - This unit, shown in Figure B-6 decodes the nanomemory bits and generates the signals for controlling the other SWI modules. The MDC also contains the counter and logic to indicate to its interpreter, data acceptance and transfer completion. There is one MDC per interpreter.

2. Device Control (DC) - The DC resolves conflicts between Interpreters trying to lock to a device and checks the lock status of any Interpreter attempting a a device operation. The DC is shown in Figure B-7, it receives requests for device operations and lock/unlock requests through the MDC. It responds by sending status signals to the MDC and control signals to the Input and Output Switch Network modules. The DC module as mechanized in the Burroughs multiprocessor provides device control for up to three interpreters. A system with five interpreters will use two DC modules.

3. Memory Control (MC) - The MC resolves conflicts between Interpreters requesting the use of the same memory module and maintains an established connection after completion of the operation until some other Interpreter requests that memory module. Figure B-8 contains a diagram indicating a typical interpreter stage in the memory control module. This stage receives requests from the MDC and a 3 bit memory module address from the interpreter. The lower section of Figure B-8 shows the memory request

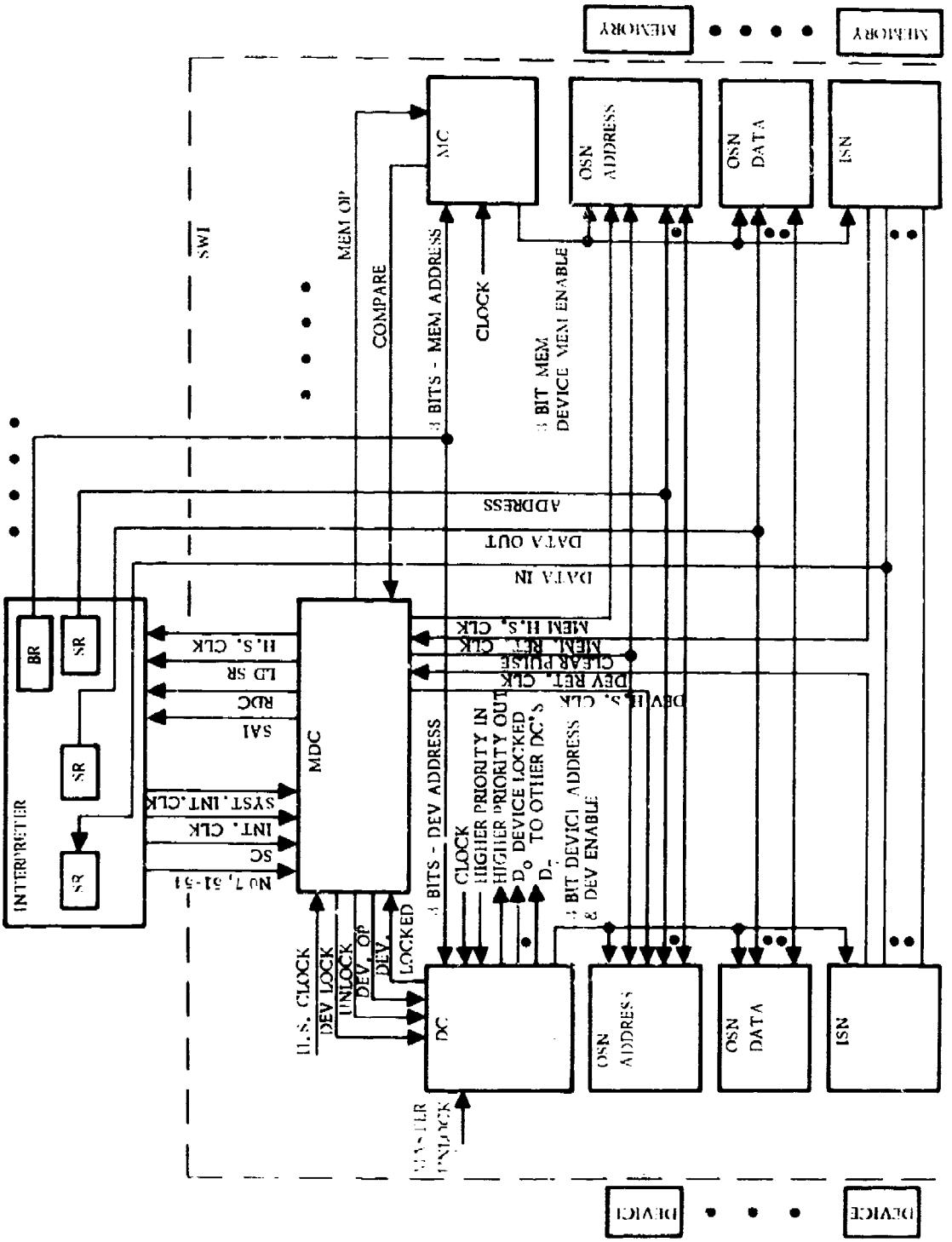


Figure B-5. SWI Interface Diagram

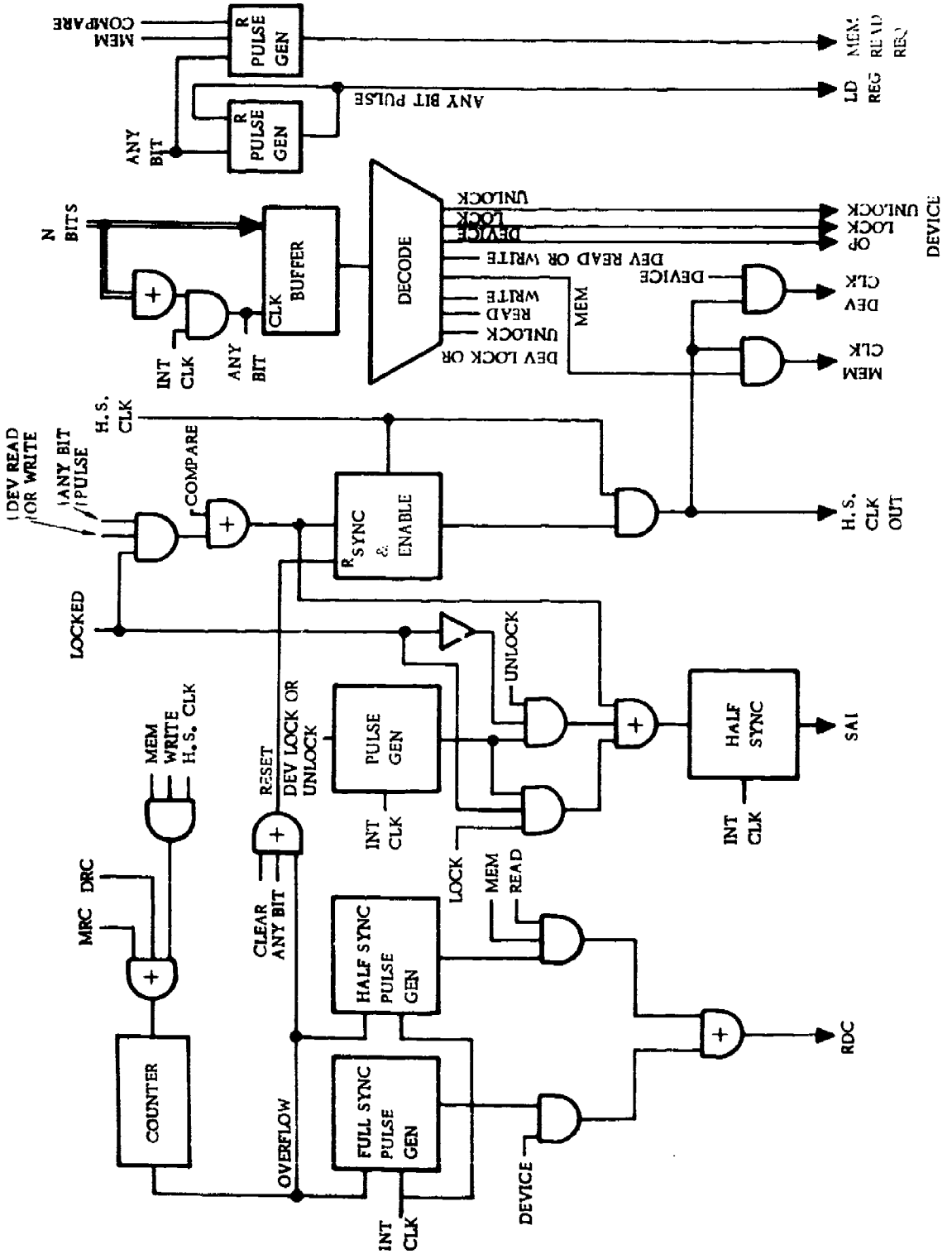


Figure B-6. MDC Block Diagram

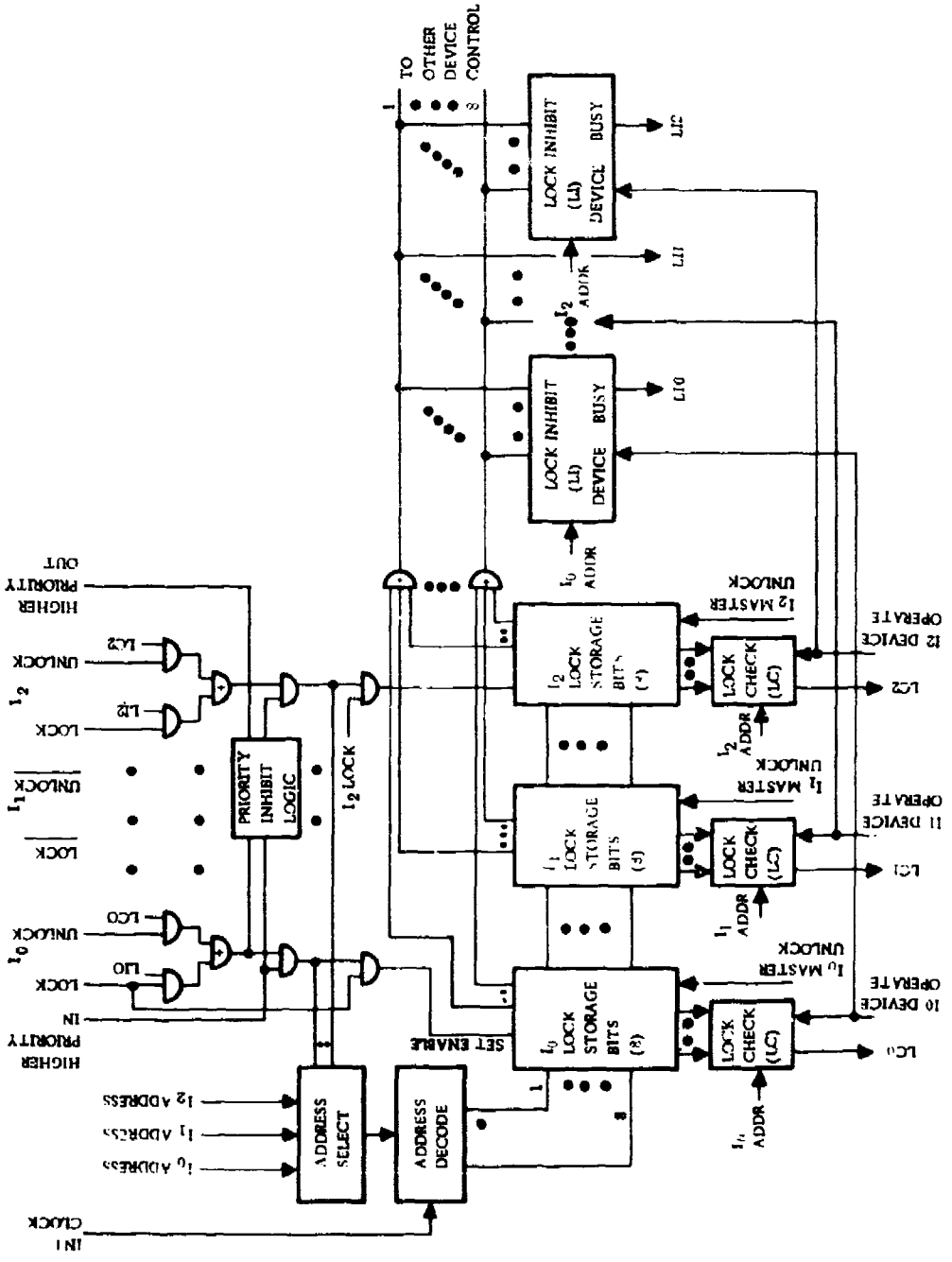


Figure B-7. 3-Channel Device Control Block Diagram

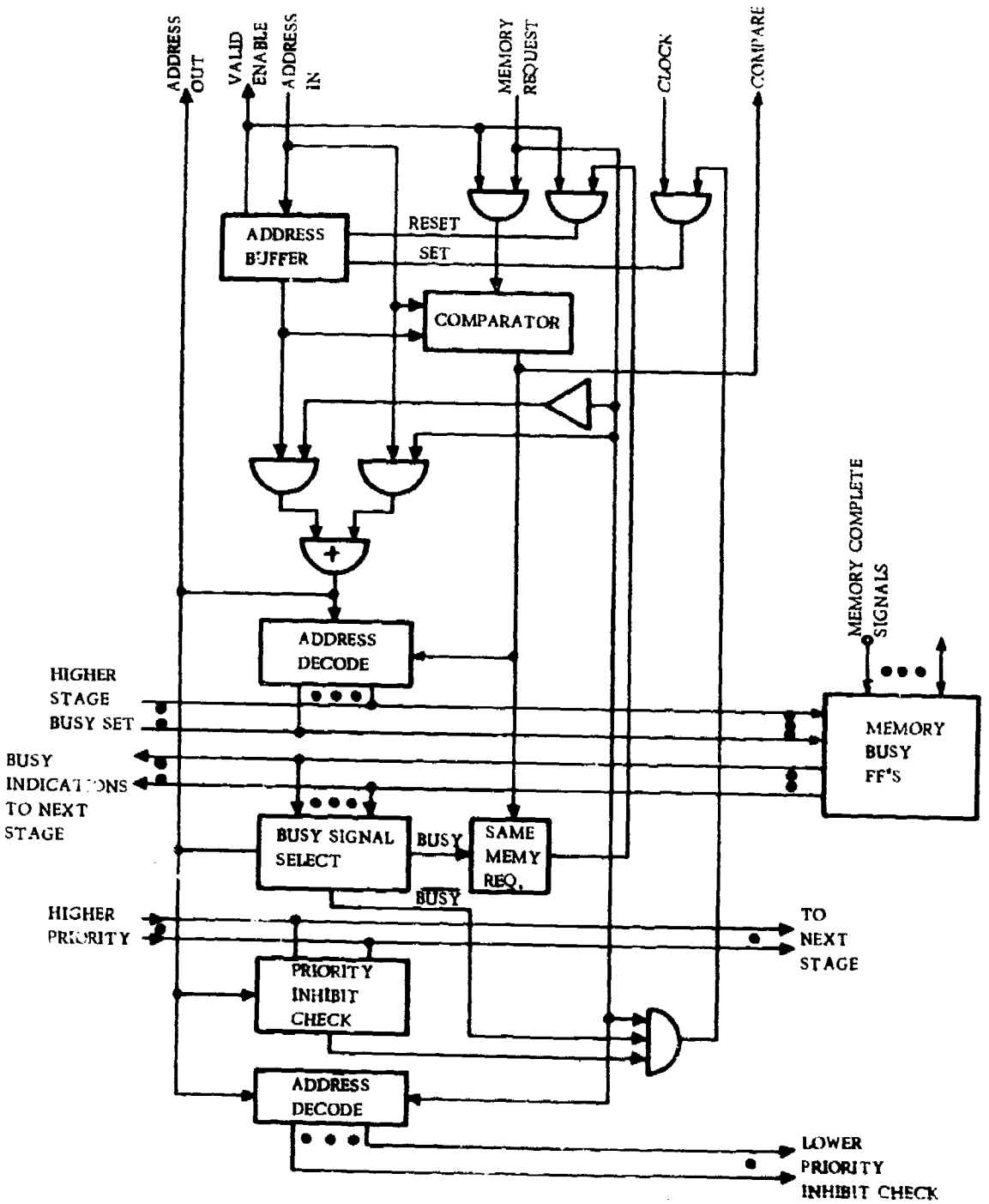


Figure B-8. Typical Stage - Memory Control

and memory busy bus that connects to the priority logic for memory request control. The Burroughs mechanization of the MC uses two modules MC0 and MC1. MC0 contains three stages as shown in Figure B-8 to provide memory control for three interpreters. MC1 contains two stages and the memory busy flip-flops.

4. Input Switch Network (ISN) - The ISN returns data from addressed devices or memory modules to the requesting interpreter (i.e., the ISN is a "Multiplexer"). As seen in Figure B-9 the ISN module provides selection for five interpreters to up to eight memories or eight devices. The ISN provides a path for 10 bits per interpreter. This path is used to provide eight data bits and a return clock, one bit is unused. The ISN module mechanized by Burroughs actually consists of two submodules, each submodule provides for 4 data bits and 1 clock bit from up to eight memories or devices to up to five interpreters. The ISN is therefore modular in terms of 4 bit bytes. The ISN is under the control of the MC or DC module.

5. Output Switch Network (OSN) - The OSN sends data, address, clock, and control from Interpreters to addressed devices or memory modules (i.e., the OSN is a "demultiplexer"). This unit is actually mechanized as two different modules. Figure B-10 shows the OSN for address output. This unit handles 4-address and 2-clock bits for five interpreters to up to eight memories on devices. The address OSN is actually mechanized from two identical submodules that provide two address bits and one clock bit each. In the Burroughs multiprocessor, the address OSN uses four address and one clock bit leaving one clock bit unused.

   The data output OSN is shown in Figure B-11. This unit provides eight bits output to up to eight memories or devices from five interpreters.

B. SWITCH INTERLOCK OPERATION AND TIMING

Controls from the Interpreter (Nanobits 51-54) are strobed into the mem dev operation register of the MDC if either the Type I microinstruction is unconditional or the selected condition is true. Controls derived from the output of this register will next load the output shift registers of the interpreter and generate one of three types of signals, depending upon the operation to be performed. Each of these types of signals will be explained.

1. Memory Operation

a. General

The first type of signal from the MDC is a "memory operation request" signal to the MC. This initiates the comparison and priority logic in the MC. When the MC has granted access by that interpreter to the memory module it was requesting, a compare signal is returned from the MC to the MDC. This will send a clear pulse to the memory interface logic through the memory OSN and will initiate the setting of SAI and the transmission of high speed clocks to the output shift registers of the interpreter and through the OSN's to the memory interface.

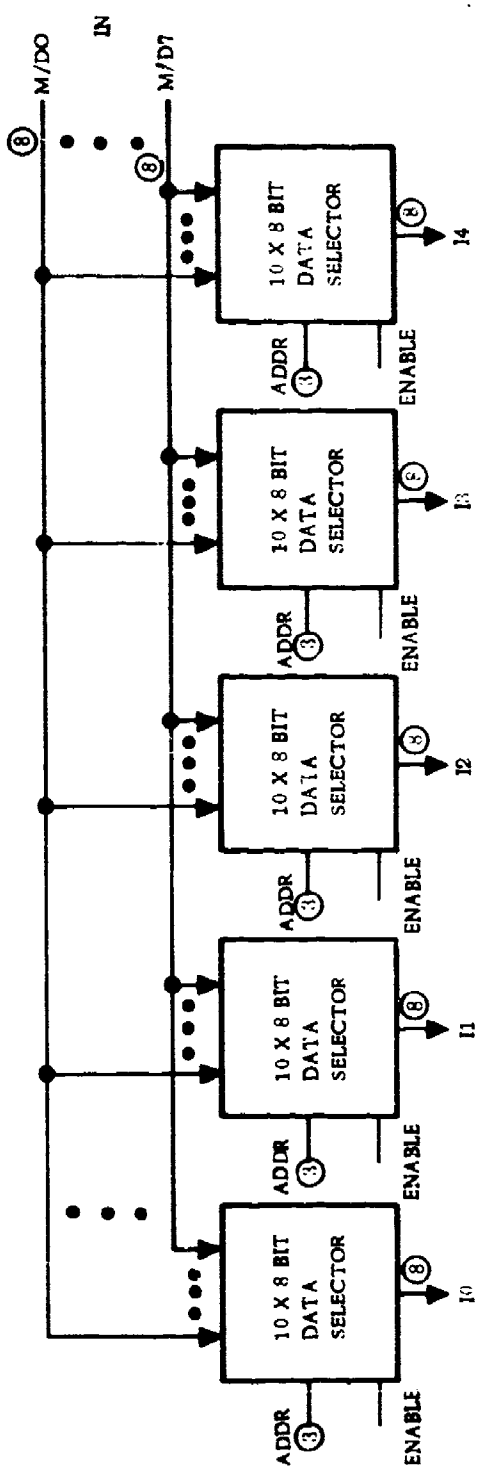


Figure B-9. 5-Channel Input Switch Network Block Diagram

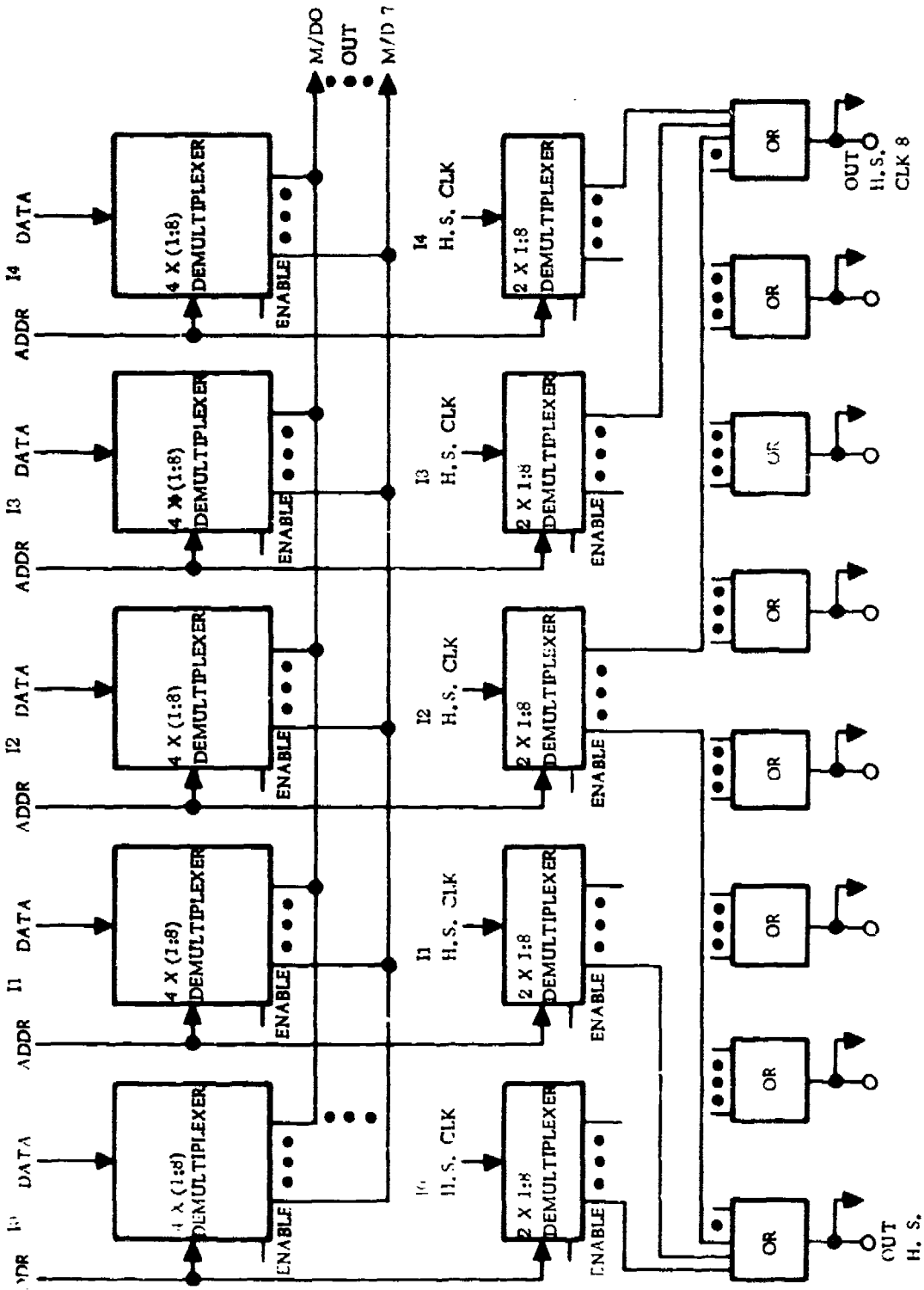


Figure B-10. 5-Channel Output Switch Network Address Block Diagram

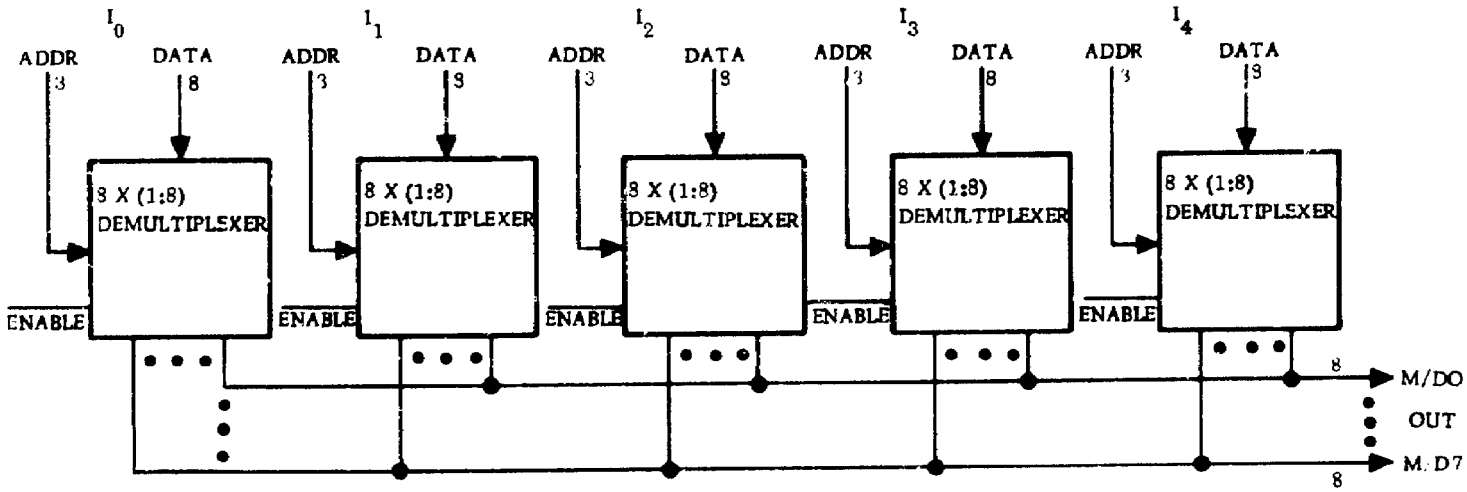


Figure B-11. 5-Channel Output Switch Network (Data) Block Diagram

In the case of a memory write, the input/output counter in the MDC will count four output high speed clocks and will then stop them.

In the case of a memory read, output high speed clocks are not counted. Instead, these high speed clocks are continually sent to the memory module interface. This interface will count four clocks coming in to it and will then initiate a memory read. Upon return of a completion signal from the memory, the memory interface will load its output shift registers and then allow four of the high speed clocks that are still coming through the OSN to clock these output shift registers and to be returned to the MDC and the interpreter with the shifted out data. The MDC counts four of these memory return clocks and will then stop the high speed output clocks and set RDC indicating that the data has been shifted into the interpreter input shift registers and is ready to be strobed into the B register.

b. Memory Groups and Interpreter Access Priority

The switch interlock module for memory connection contains a group of 8 ports for connecting memory modules to interpreters. Each port provides MR and MW. Concurrent access to all memories in a group by different Interpreters is permitted. Interpreters have fixed priority for access to all modules of a group.

Conflicts in access to the same module are resolved in favor of the Interpreter that last accessed the module, otherwise the highest priority requesting Interpreter. Once access is granted it continues until that memory operation is complete. When one access is complete, the highest priority request is honored from those Interpreters then in contention. The Interpreter completing access is not able to compete again for one clock. Thus the two highest priority Interpreters are assured of access. Lower priority Interpreters may have their access rate significantly curtailed.

The switch interlock "remembers" the prior connection of each memory module to some Interpreter. If the next request is also from the remembered Interpreter, the new connection is made with less delay, since no priority resolution need take place.

c. Memory Use Sequence

The sequence of operations necessary to access S-memory is simple in single Interpreter systems where no conflict in access can exist. In such cases once the address setup is complete (as is the MIR for write), the memory read (or write) can be initiated. After a suitable time the data from memory can be accessed via BEX or BBE. In the presence of conflict potential, the following control sequence should be used. This sequence is recommended for sytems without a switch interlock as well.

1. The S-memory address should be in the selected base register and MAR.

2. Memory Read

2.1 A test of RDC should be included in some prior instruction. By convention this should be the previous memory read (or device read or write by request). A test of SAI should be included if address register changes are required before the RDC is returned, or if confirmation of access to the switch interlock is desired.

2.2 The memory read can occur the instruction after the address is (unconditionally) loaded into MAR1 or MAR2.

2.3 A SAI is returned when the switch interlock has accepted the address and the memory is connected to the requesting Interpreter through the switch interlock.

2.4 A group of intervening instructions can be issued, depending on the relative speeds of the Interpreter clock and the S-memory. Once SAI is set and tested, these instructions may change the address registers or even include device read or write operation on demand.

2.5 A RDC (read complete) signal is returned when data will become available for entry into the Interpreter the following clock.

2.6 If no intervening device or memory reads occur, BEX may be repeated each time receiving the data in XDI non-destructively.

3. Memory Write

3.1 The data to be written should be in MIR.

3.2 The address should be in the selected base register and MAR.

3.3 The memory read can occur the instruction after both the address and data have the desired values.

3.4 Return of SAI indicates that the memory is connected and therefore the address and data have been accepted in the XDA and XDO buffer registers respectively, and thus the address registers and MIR may be subsequently changed.

3.5 It is possible that the memory is still in its memory cycle, but if so, no other access will be granted to that memory module.

2. Device Lock and Unlock

The second type of signal emanating from the MDC is a device lock or device unlock request sent to the DC. After the DC has accomplished this, a signal is returned to the MDC in order to set SAI and the operation is complete.

The switch interlock module for device connection contains a group of eight ports for connecting devices to up to five interperters. DL, DR, DW, and DU are provided for each port. Priority order for resolving concurrent requests by Interpreters for DL or DU is fixed within each group.

Conflicts in DL and DU requests may occur. The DL request from the highest priority requesting Interpreter is honored over a co-occurring request for the same device from any lower priority Interpreter. Concurrent DL requests for different devices in the same group cause the lower priority request to incur a one clock delay in achieving the DL or DU, and in return of SAI for each higher priority request. Consequently DL or DU requests from Interpreters other than the highest priority may be

arbitrarily delayed. The earliest confirming SAI response occurs two instructions after issue of the DL or DU. If SAI is true, then the DL or DU was successful. If SAI is false, then it means that the DL or DU is not yet successful. The design justification for this potential arbitrary delay is that DL or DU are infrequent events for which arbitrary delay is of little consequence.

Provision for conscious control of this timing is provided (and recommended) by use of Global Conditional Bit 2 to protect DL and DU attempts by more than one Interpreter at a time.

3. Device Read and Write

a. General

The third type of signal from the MDC occurs for device reads or writes and is sent to the DC to check the lock status of the device being addressed by the BR1/BR2 of the interpreter before proceeding. After it is confirmed that the device is locked, the DC returns a locked signal to the MDC. This will have the same effect as when a memory module is obtained, i.e., a clear pulse is sent to the device interface logic through the device OSN and initiates the setting of SAI and transmission of high speed clocks to the output shift register of the interpreter and through the OSN's to the device interface.

However, the distinction made between memory reads and memory writes is not made for devices. Both cases act like a memory read; i.e., for a device write the MDC does not stop the outgoing high speed clock after four clocks and indeed does not even count them. In both cases the device interface counts four clocks coming in to it and then stops accepting high speed clocks. In the case of a read, the device interface waits for some kind of "data available" signal from the device which it will use to load its output shift registers and to allow four high speed clocks which are still arriving from the OSN to clock these output shift registers and to be returned to the MDC and the interpreter with the data. The MDC, as for memory reads, counts return clocks and will set RDC.

In the case of a write, the response is very dependent upon the particular device being interfaced. In the case of a card reader, Burroughs sent back the next four high speed clocks to the Interpreter. In the case of a printer, Burroughs used a signal saying the last character was accepted by the printer to cause the device interface to allow return clocks. The four return clocks are counted by the MDC and used as a means of saying that the device accepted the data sent out.

b. Duration of Device Operations

The duration of a DL request depends on its success. If successful the lock occurs concurrent with the following instruction, at the end of which SAI is set true. Thus SAI is available for test in the second following instruction. If false at this time, the DL request continues while other work may be in progress so long as neither the device identification changes nor another memory or device operation is initiated. When the previously issued DL is successful, SAI will be returned.

Device reads or writes are only completed with devices locked to the Interpreter issuing the DR or DW. Depending on the device address, a DR or DW may be on demand or by request.

1. On Demand. DR and DW provide immediate data exchange. The duration of DR or DW on demand is one instruction after issue. Confirmation of completion may be checked by SAI being true the second instruction after issue. If SAI is false, the device was not locked to the requesting Interpreter.

2. By Request. DR and DW provide data exchange when the device is ready. The duration of DR or DW by request is determined by the device and is signalled to the requesting Interpreter by the return of RDC - "Request of device complete." As with DR and DW on demand, SAI is returned by the second instruction after issue, and indicates that the device is locked to the requesting Interpreter.

The duration of DU is one instruction after issue, unless conflicts from DL or DU requests by other Interpreters occur as indicated above. SAI will be returned only if the device had been locked to the requesting interpreter. SAI is available for test in the second instruction after issue if no conflicts arise. Any conflict with other DL or DU in the same group can cause delay.

c. Device Use Sequence

The sequence of device operations necessary for an Interpreter to use a device is as follows:

1. A test of "IF SAI" should be included in some instruction to reset it. This usually can be in the instruction with the unconditional device operation.

2. Device Lock Request: The data in the indicated base register (and possibly MAR) is used as the device identification. On the second following instruction, SAI may be tested.

2.1 If true, then the device lock was successful.

2.2 If false, then the device lock was unsuccessful. The request remains in progress while other instructions not chaning the device identification or issuing other memory or device operation may be executed. The DL request is terminated by the first of the following actions:

(a) The Interpreter initiates another memory or device operation.

(b) The Interpreter changes the device identification.

(c) The device becomes available and sets SAI. All co-occurring actions are valid. Should (a) and (c) co-occur, SAI refers to the DL in the following instruction and should be tested. Then in the next instruction thereafter SAI refers to the new memory or device operation. Should termination by (b) occur without co-occurrence of (c), the new device identification applies to the DL still in progress, and the path for SAI return is diverted to the newly identified device (if there is one so identified) without reissue of another DL.

3. Once the desired device is locked to the Interpreter. a sequence of one or more data exchanges may be initiated using the following. Assume for simplicity that adequate bandwidth connection is provided so that data transfer is completed in one clock. Otherwise add an appropriate number of clock times to the discussion.

4. Device Write: The data in the indicated base register (and possibly MAR) is used to specify the device, and the data in the MIR provides the information to be written to the device. The second instruction after the device write, SAI may be tested. If true, the Interpreter is locked to the device, and data in the MIR has been accepted by the XDO register, and so the MIR may subsequently be changed. If false, the Interpreter was not locked to the requesting device.

4.1 On Demand: The device is immediately ready to accept input data from the Interpreter. Consequently the SAI need not be checked, and the MIR or device identification could even be changed in the instruction after the DW.

4.2 By Request: The device provides an RDC when it has completed the requested write. The SAI also indicates that the MIR data has been accepted in the switch interlock. Similar to DL, the request continues until the first of the corresponding three actions.

(a) The Interpreter initiates another memory or device operation.

(b) The Interpreter changes the device identification.

(c) The DW is completed and sets RDC. All co-occurring actions are valid. Should (a) and (c) co-occur, SAI refers to the DW in the following instruction and should be tested. In the next following instruction SAI then refers to the new memory or device operation. Should (b) not co-occur with (c), then the DW in progress is diverted to apply to the new device identification without reissue of another DL.

4.3 Separate device identifications are required if the same device is to be read both on demand and by request (some distinguishing bit).

5. Device Read: The data in the specified base register and MAR is used to specify the device. The second instruction after the device read, SAI may be tested. If true, the Interpreter is locked to the device; otherwise not.

5.1 On Demand: The device output register is assumed immediately able to be read on demand (possibly some part of the resulting data indicates validity). The data requested is available for Interpreter access the clock after Phase 1. Thus BEX or BBE may be included in the same instruction as the device read. The SAI need not be checked, and the device identification may be changed in the instruction after the DR (so long as the address is still not required for a prior memory operation or device read by request).

5.2 By Request: The device provides a RDC after the device read request when it has sent the desired data from its output register. Thus the same instruction that finds RDC true may include BEX. RDC should be reset by testing prior to use for device read by request (usually as part of the prior instruction using BEX).

5.3 Separate device identifications are required if the same device is to be read both on demand and by request.

6. Device Unlock: When use of the device is completed the lock should be terminated by issuing a device unlock. An SAI is returned if the issuing Interpreter was locked to the device. An attempt to unlock a device that is not locked to the Interpreter will not return SAI. SAI is available for test at earliest the second instruction after the device unlock.

5. SOFTWARE

A. INTRODUCTION

The basic philosophy behind the multiprocessor executive (Ref B-6) is that it operates independent of any particular interpreter. The executive is structured such that no interpreter operates as a hardcore or has control of the system. In effect the executive functions "float" among all interpreters resulting in a distributed executive.

Figure B-12 presents a grossly simplified diagram of the operation of the system. The hardware portion of the system is depicted as Interpreters and Main Memory. In particular the microprogram memory of each interpreter is shown. The distribution of software elements among the microprogram memory and main memory is shown in this figure. Each microprogram memory contains a portion that is permanent, namely the locator and allocator functions. In addition a microprogram and various executive modules are contained in the microprogram memory that may vary depending on the needs of each task.

Tasks constitute particular jobs or functions desired to be performed on the computer system. Each task has its own work area in main memory and contains a microprogram, S program, and data area in main memory. The microprogram is loaded into an interpreter's microprogram memory when the task is selected for execution. A number of executive modules are also stored in main memory. A task may select and execute these executive modules by loading them into microprogram memory. One of the executive modules is the scheduler. This module is run after a task is ended and uses the executive tables to define the next task to be run. Each interpreter uses the same executive functions and performs its own scheduling and other executive functions.

B. TASK OPERATION

1. Task Work Area

Once a task is selected by an interpreter for execution, its work area, located in main memory (pointed to by the task table in the executive table), defines where and how to run the task. A task is initiated or continued via its work area (Figure B-13).

The Work area contains a state vector which defines the state (all the registers) of the 'S' machine (Figure B-14). The state vector also defines where and which microcode must be in microprogram memory for the task to run. In addition to the pointer to the microprogram location in main memory, the state vector contains the address of where to start the microprogram.

A program reference table is also part of the task work area. This table contains pointers to the 'S' programs and data areas needed for the task. The table also contains other information such as whether the 'S' programs are in main memory or in mass memory. The use of this table as part of the task work area is shown in Figure B-15. The task table is part of the executive table that the scheduler uses and will be explained later.

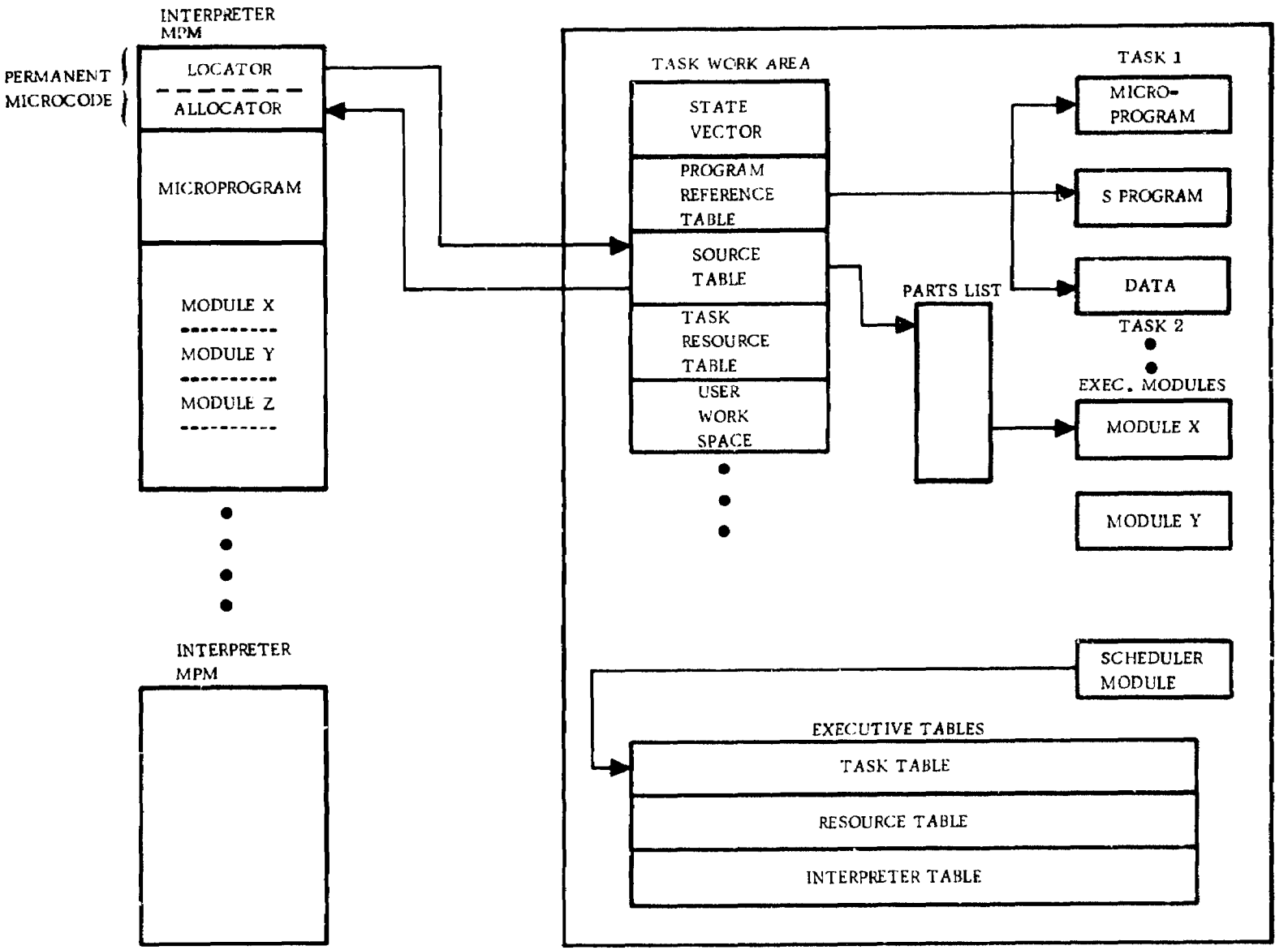


Figure B-12. Software Operation

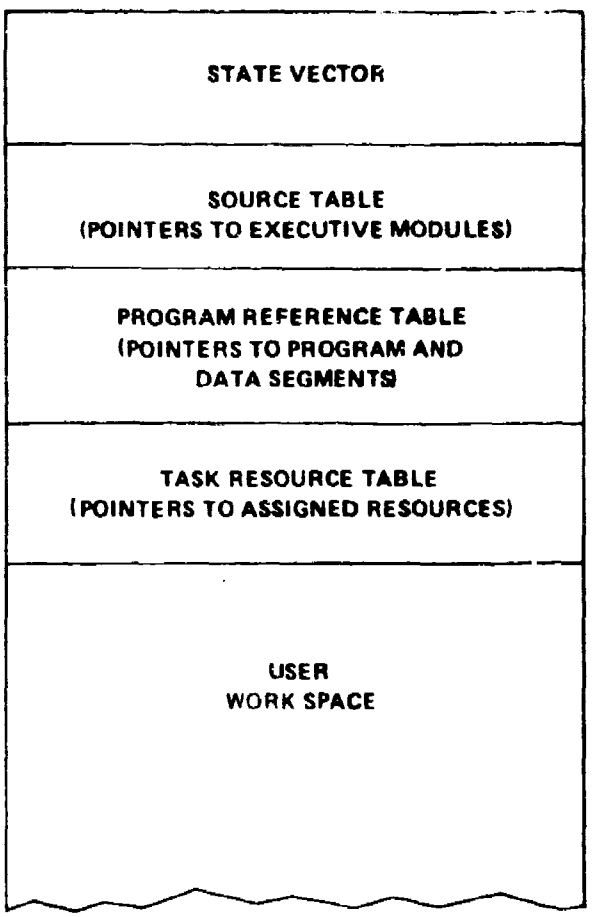


Figure B-13. Contents of the Work Area

POINTER TO USER WORK SPACE		POINTER TO LAST USED WORD IN AREA
LOCATION OF POINTER TO MICROPROGRAM		ADDRESS TO START MICROPROGRAM
TASK TABLE ENTRY NO.	ASSOCIATED TASKS WHOSE RR BITS MUST BE SET	
"S" MACHINE REGISTERS (DEFINED BY MICROPROGRAM) (PCR, BASE REGISTERS, ACCUMULATORS, INDEX REGISTERS, ETC.)		
TASK TABLE ENTRY READY-TO-RUN BITS AND RESOURCES		
GLOBAL LUC		

Figure B-14. State Vector

A source table is also contained in the task work area. This table contains pointers to the executive modules that are needed to run this task. The source table also contains information such as whether the executive module is in microprogram memory or in main mamory as shown below:

LUC	Size	Location in Parts List	Location in Microprogram Memory or Location of Allocator

A task resource table in the work area defines which resources have been assigned to a task and which resources are unavailable. The user work space in the task work area is used to maintain a local operating environment for a module so that the modules may be written in reentrant code.

2. Microprogram Memory

The microprogram memory of an interpreter contains three separate areas (variable bounds): the locator/allocator, the microprogram, and the executive modules.

The locator/allocator is permanent microcode that is invariant of the task being run. In running a task various executive modules may be called upon to execute certain functions such as I/O from a peripheral, subroutines, etc. These modules are executed from microprogram memory. The task calls the locator to execute an executive module (Figure B-16). The locator saves the return address in the task and restores it after the module is executed. The locator uses the index supplied by the task to select the proper entry in the source table of the task's work area to locate the module, Figure B-17 depicts how a module is located. If the source table indicates the module

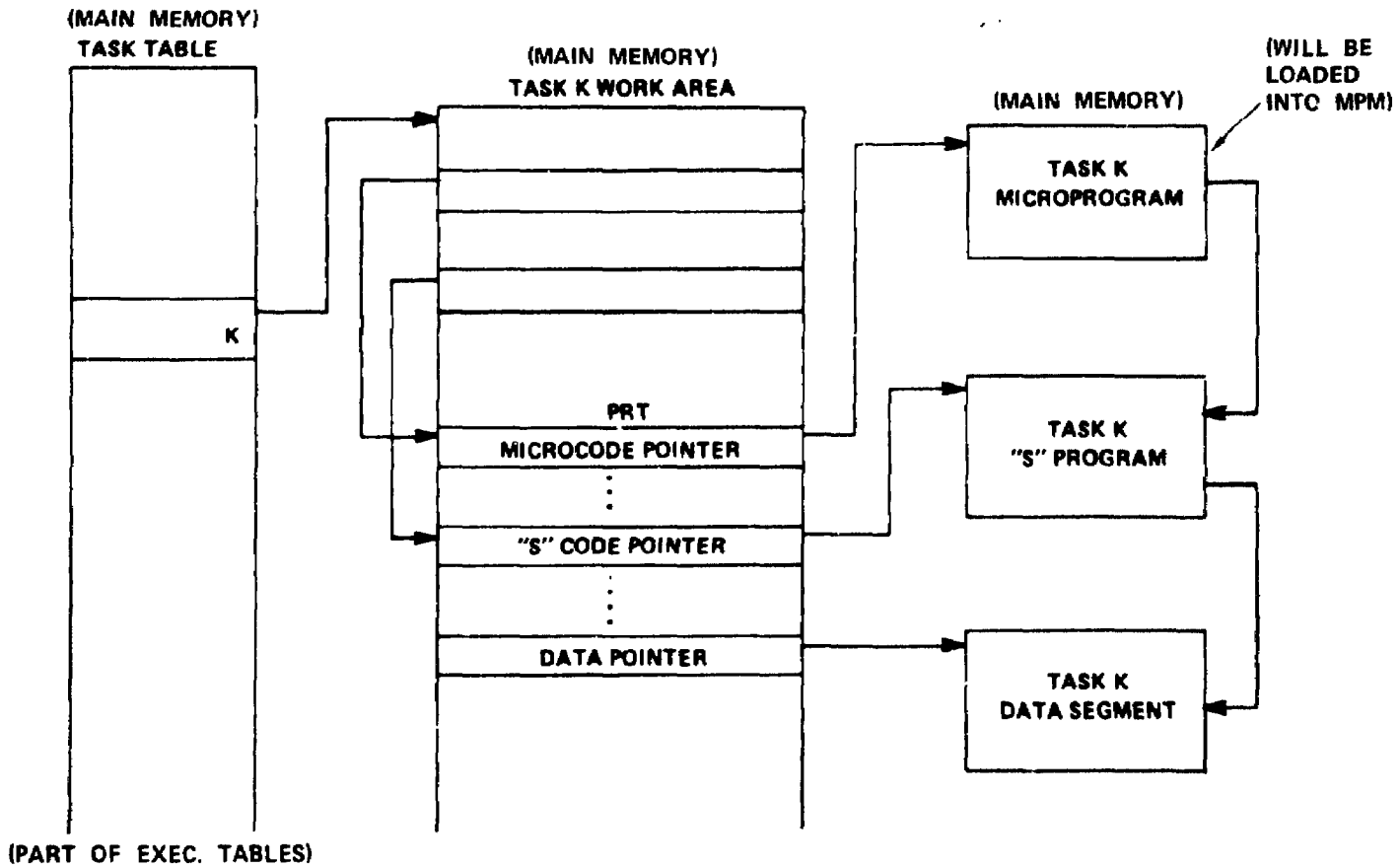


Figure B-15. Using the Program Reference Table

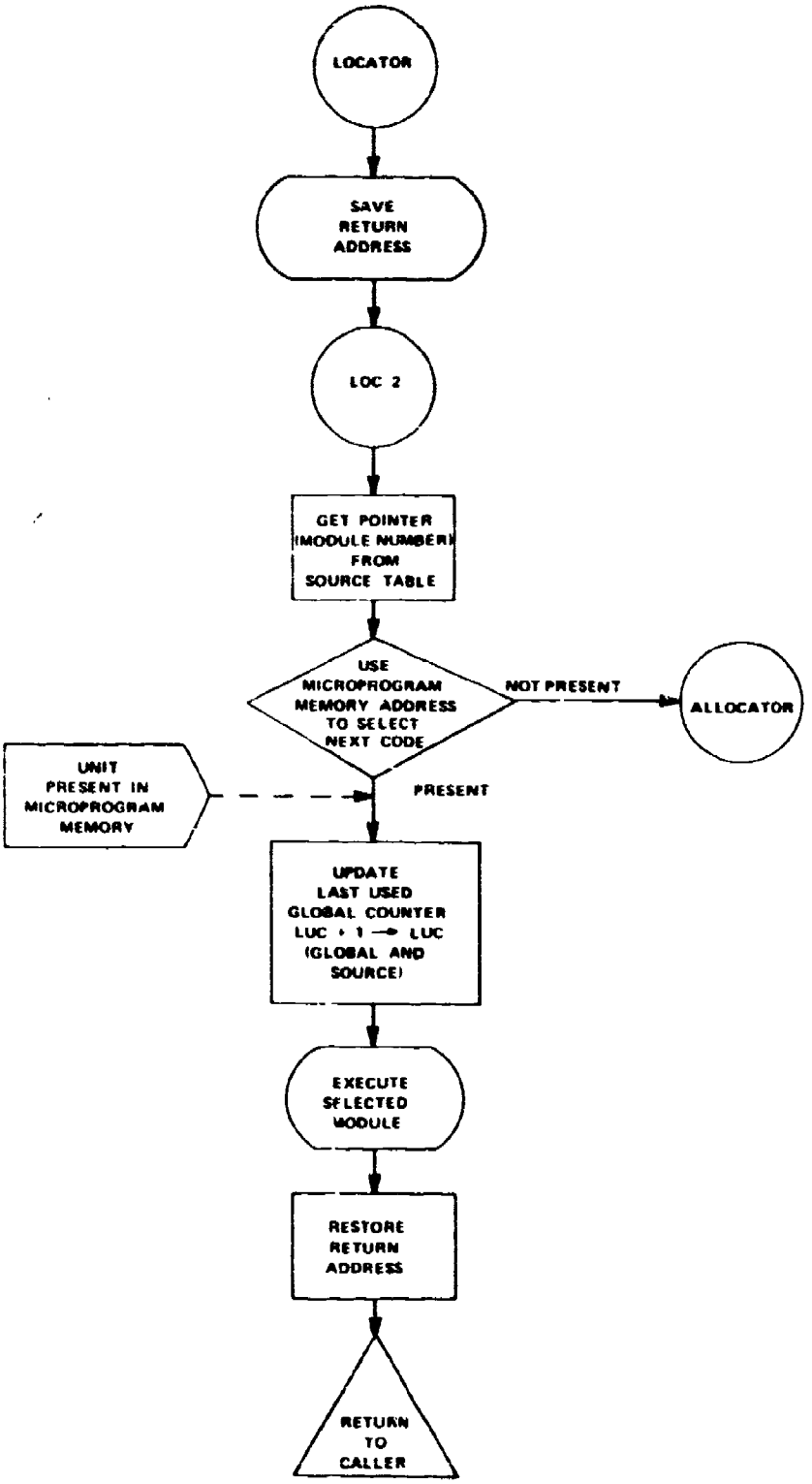


Figure B-16. Locator

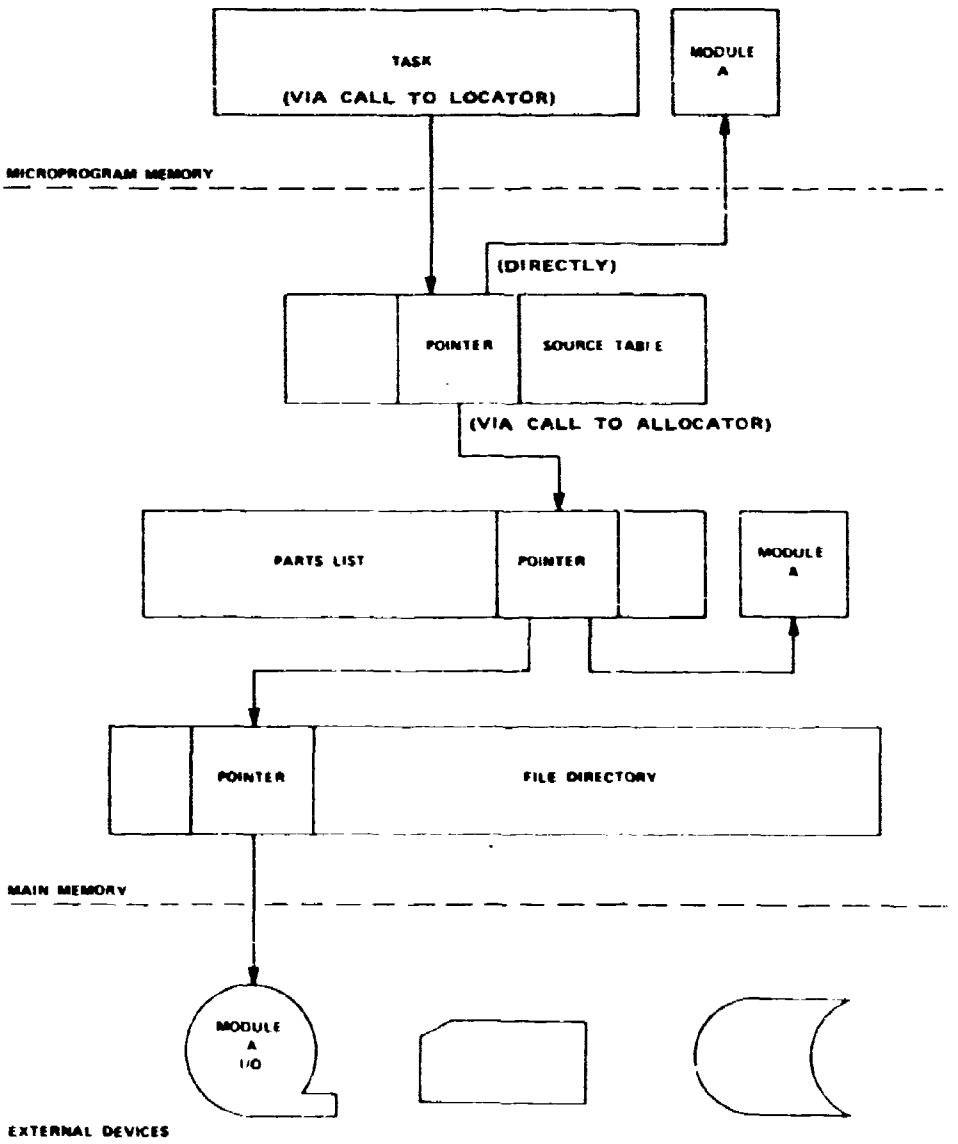


Figure B-17. Locating a Module Through Storage Hierarchy

is in microprogram memory, it will contain a pointer to the module. If it is not in microprogram memory, it will point to the allocator. The allocator then uses the entry in the source table to point to a Parts List Table (this is a master table with one per system) which will contain a pointer to the module in main memory.

If the module were not in main memory, the Parts List would point to a file directory that would point to a location on some peripheral. The allocator function is used to find space and copy an executive module into the microprogram memory to execute it (Figure B-18). The allocator function may also have to deallocate some modules from the microprogram memory if enough free space is not available therein. Modules are deallocated (overlayed) based upon their relative use by the task being executed.

C. EXECUTIVE OPERATION

There are a number of modules that perform executive functions. Two modules that perform tasks basic to operation of the computer system are the scheduler and end task modules. The scheduler module scans the task table to determine which task is to be executed next. The task table (Figure B-19) contains bits defining if a task is presently being run, if it is ready to run, its S machine ID. its work area pointer, etc.

Each interpreter is assigned a global bit. Only one interpreter can have its associated global bit set at any one time. Simultaneous requests to set a global bit are resolved in favor of the highest priority interpreter. An interpreter must set its global bit before changing any bit of a task in the task table, (this is shown in Figure B-20). This prevents conflicts between interpreters in simultaneously attempting to make changes in the executive tables.

Another executive table is the resource table (Figure B-21). This table defines if a resource is being used and contains a link to a resource waiting list.

An additional executive table, the interpreter table (Figure B-22), contains information defining the status of each interpreter in the system. The interpreter table defines if an interpreter is busy, what task is being run on , a time check on the interpreter for failure detection (an interpreter has to repo. n to the table within a specified time interval after starting a task), and a communication area for sending messages between interpreters.

The end task module unschedules a task. This module updates system tables, releases resources, and checks the interpreter tables to check on the operation of the other interpreters.

The use of the end task and scheduler modules in the cyclic flow of execution of tasks is shown in Figure B-23. The scheduler module selects a task to run and allocates resources as needed. Execution is then turned over to the task which will select and run any executive modules needed. The task must also report in periodically to its interpreter table. When the task is to be ended the end or suspend task module performs the functions illustrated whereupon it then calls upon the scheduler module to initiate the process over again.

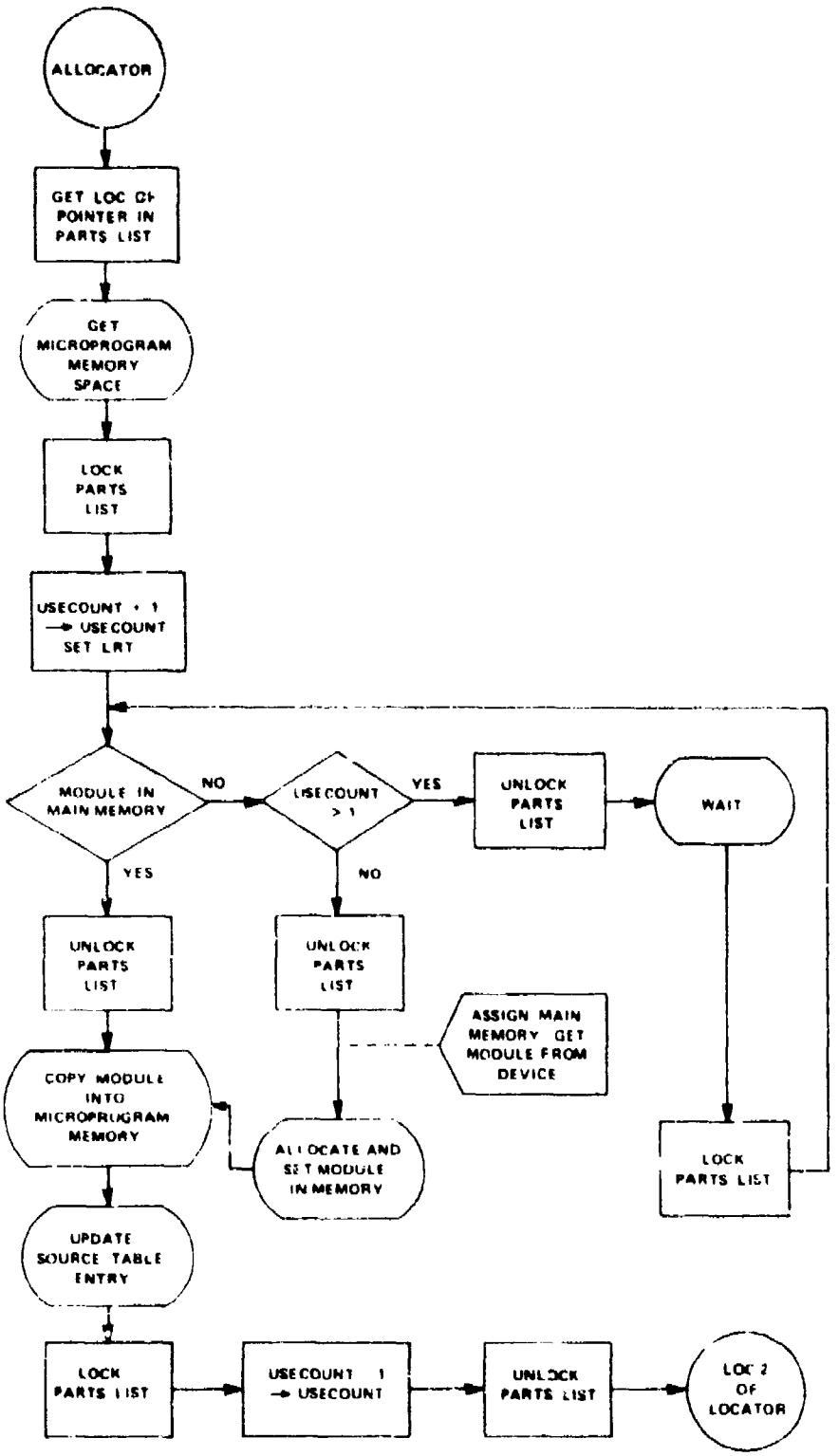


Figure B-18. Allocator

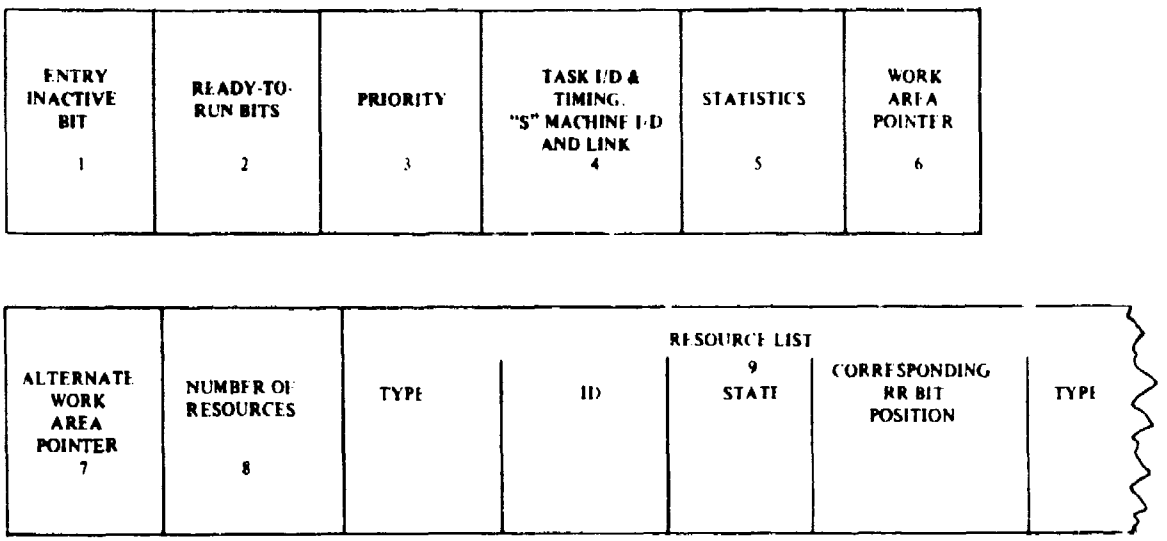


Figure B-19. Task Table Entry.

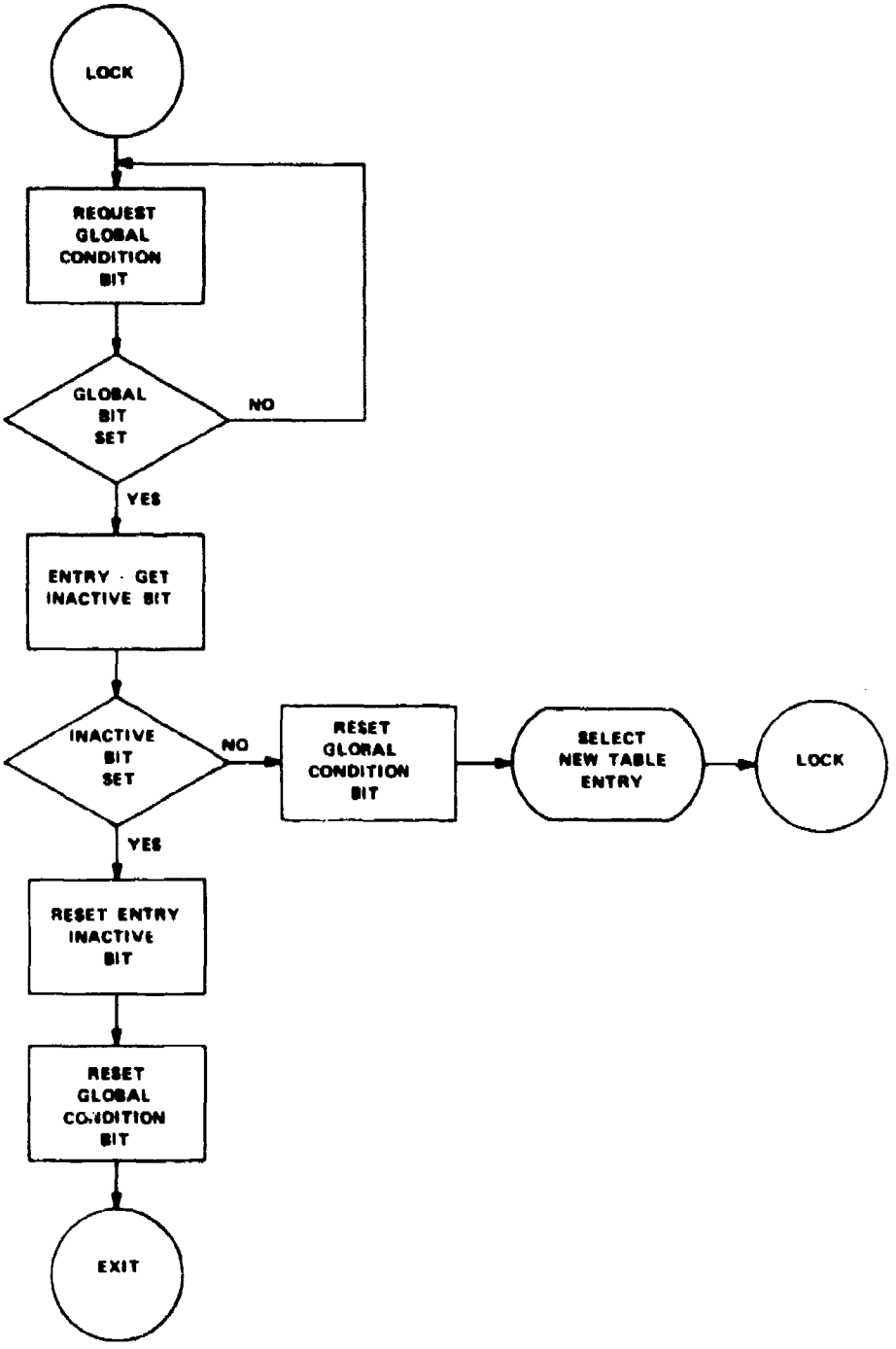


Figure B-20. Locking

ENTRY INACTIVE BIT	TYPE	ID	STATE	TASK	TASK PRIORITY	TYPE LINK	WAITING LIST LINK	HIGHEST PRIORITY WAITING
1	2	3	4	5	6	7	8	9

Figure B-21. Resource Table Entry

ENTRY INACTIVE BIT	INTERPRETER DOWN	RUN DIAGNOSTICS	TASK NUMBER	START TIME	WAIT TIME	TIME NEXT REPORT DUE	COMMUNICATION AREA
1	2	3	4	5	6	7	8

Figure B-22. Interpreter Table Entry

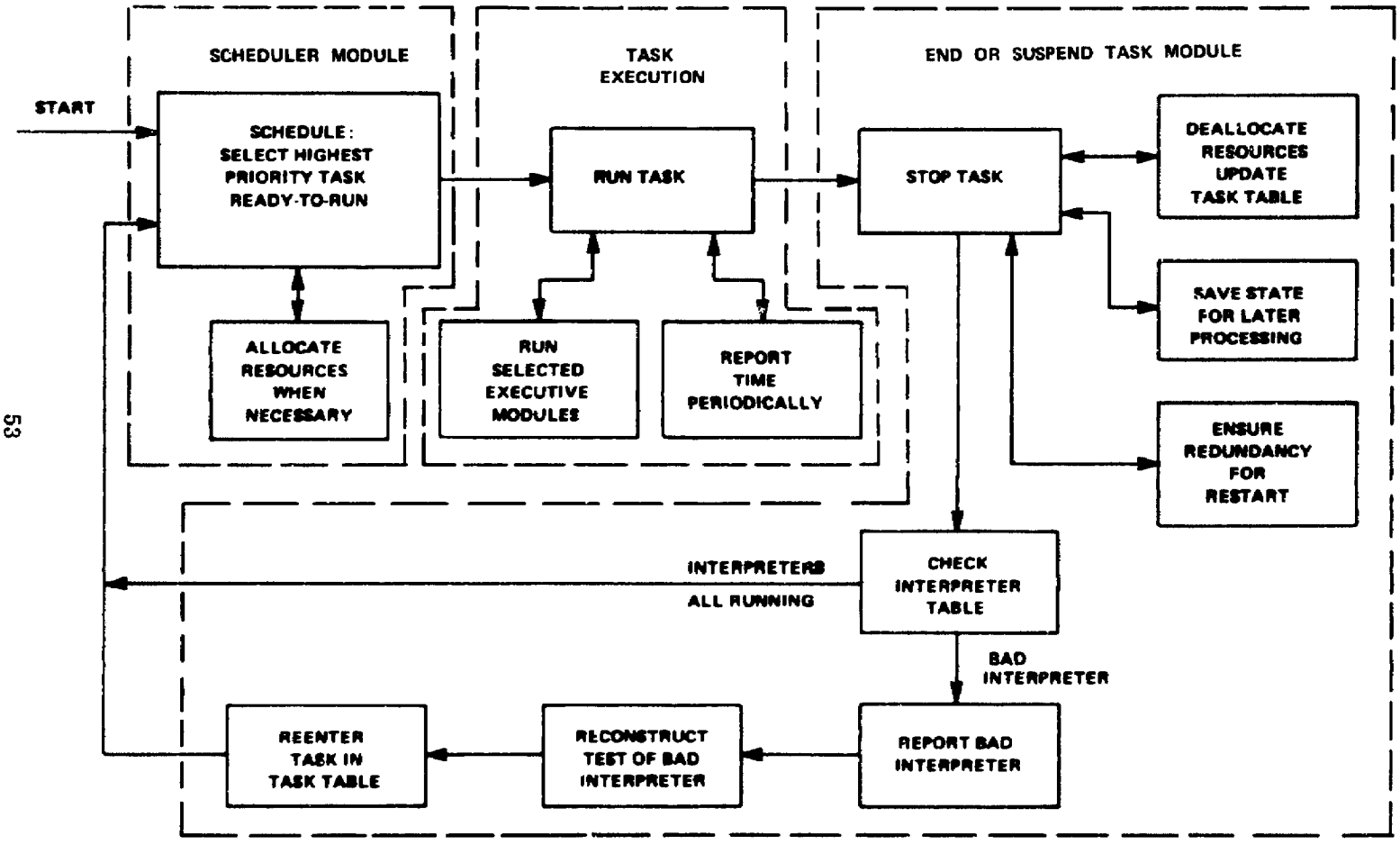


Figure B-23. Cyclic Flow of the System

REFERENCES

B-1. Davis, R. L., C. M. Campbell, S. Zucker; Aerospace Multiprocessor Interim Report, Burroughs Corporation, Feb 1972

B-2. Wehr, K. C., Technical Summary of the Interpreter-Based System, Burroughs Corporation, Jan 1971

B-3. Davis, R. L., and S. Zucker; Structure of a Multiprocessor Using Microprogrammable Building Blocks, NAECON '71 Record, pp 186-200.

B-4. Davis, R.L., and S. Zucker, C. M. Campbell, A. Building Block Approach to Multiprocessing, 1972 Spring Joint Computer Conference, pp 685-703.

B-5. Reigel, E. W., U. Faber, D. A. Fisher, The Interpreter - A Microprogrammable Building Block System, 1972 SJCC, pp 705-723.

B-6. Zucker, S., Aerospace Multiprocessor Executive, Burroughs Corp, Paoli, Pa., Technical Report AFAL-TR-72-144, April 1972.

DISTRIBUTION LIST

Contract F33615-72-C-1973

Address	No. of Copies
WPAFB ACTIVITIES	
AFAL/TSR WPAFB OH 45433	1
AFAL/AAM (Mr. J. Camp) WPAFB OH 45433	18
AFIT (Library) WPAFB OH 45433	2
ASD/YHEV (Mr. Jim Hutson) WPAFB OH 45433	2
2750ABW/SSL WPAFB OH 45433	1
OTHER ACTIVITIES	
HQ USAF/SAMID Wash DC 20330	1
AU Library Maxwell AFB AL 36112	1
Director Naval Research Lab Wash DC 20390	1
Commanding Officer Naval Avionics Facility 21st and Arlington Ave Indianapolis IN 46218	1
US Army Electronics R&D Lab Attn: Dr. H. Jacobs Ft Monmouth NJ 07703	1
Director, NSA R-13 Ft George Meade MD 20755	1

DDC 2
Cameron Station
Alexandria VA 22314

INDUSTRY

Control Data Corp 1
4130 Linden Ave
Dayton OH 45432

Hughes Aircraft Co 1
Aerospace Group
Culver City CA 90230

Honeywell 1
Military Products Group
2314 Standly Ave
Dayton OH 45404

IBM Corp 1
33 West First St
Dayton OH 45402

RCA 1
Aerospace Systems Division
Box 588
Burlington MA 01801

McDonnell Douglas Corp 1
333 West First St
Dayton OH 45402

Raytheon 1
333 West First St
Dayton, Ohio 45402

Westinghouse Electric Corp 1
Aerospace Division
Friendship International Airport
Box 746
Baltimore MD 21203

Litton Systems, Inc. 1
Guidance & COntrol System Division
5500
Canoga Ave
Woodland Hills CA 91364

Texas Instruments, Inc. 1
Equipment Group
Suite 205
3300 South Dixie Drive
Dayton OH 45439

General Electric Co 1
Aerospace & Defense Sales & Service
3430
South Dixie
Dayton OH 45439

Univac 1
Defence Systems Division
333 West First St
Dayton OH 45402

Burroughs Corp 1
Federal & Special Systems Group
Attn: D.F. Sullivan
Paoli PA 19301

Boeing Computing Systems 1
Attn: J.F. Cramer
8R-39 Mail Stop
Box 3707
Seattle WA 98124

Singer-Kearfott Division 1
Attn: M.G. Page
33 West First St
Dayton OH 45402

The Garrett Corp 1
333 West First St
Dayton OH 45402

Grumman Aircraft 1
333 West First St
Dayton OH 45402

Northrop Corp 1
379 West First St
Dayton OH 45402

Department of Transportation 1
Transportation Systems Center
Attn: Mr. G. Y. Wang
Cambridge, Mass.

National Aeronautics and Space Administration
Langley Research Center
Attn: Mr. L. Spencer
Hampton, Virginia 23365

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Autonetics Division of Rockwell International
3370 E. Mraloma Ave, Anaheim, Ca. 92803

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
Avionics Processor Controller STudy, Volume 3, Multiprocessor Description

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Report July 1972 - June 1973

5. AUTHOR(S) *(First name, middle initial, last name)*
L. J. Koczela

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 30, 1973	59	6

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F33615-72-C-1973 b. PROJECT NO.	C72-812/201, Vol 3
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* AFAL-TR-73-203, Vol. 3
d.	

10. DISTRIBUTION STATEMENT
Distribution limited to U.S. Government Agencies only; test and evaluation results reported; February 1972. Other requests for this document must be referred to Air Force Avionics Laboratory (AAM), Wright-Patterson Air Force Base, OH 45433.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	AFAL/AAM WPAFB, Ohio 45433

13. ABSTRACT

This volume presents a detailed description of the Burroughs Multiprocessor. The descriptive material of the multiprocessor was scattered through several reports. The purpose of this volume is to extract the appropriate material from these reports and present the available material, upon which the study was based, in one unified report.

DD FORM 1 NOV 1473

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Multiprocessing Computer Archituecture Computer Organization Microprogramming						